ЗАТВЕРДЖЕНО

Науково-методичною радою
Державного університету
«Житомирська політехніка»

протокол від 21 травня 2020 р.
№1

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ
для проведення практичних (лабораторних) занять з навчальної дисципліни «Цифрова техніка та ПЛІС Ч2»

для здобувачів вищої освіти освітнього ступеня «бакалавр»
спеціальності 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Телекомунікації та радіотехніка»
факультет інформаційно-комп’ютерних технологій
кафедра комп'ютерних технологій у медицині та телекомунікаціях

Рекомендовано на засіданні
кафедри комп'ютерних технологій
у медицині та телекомунікаціях
28 серпня 2023 р., протокол №7

Розробник: к.т.н., доцент кафедри комп'ютерних технологій у медицині та телекомунікаціях ЦИПОРЕНКО Віталій

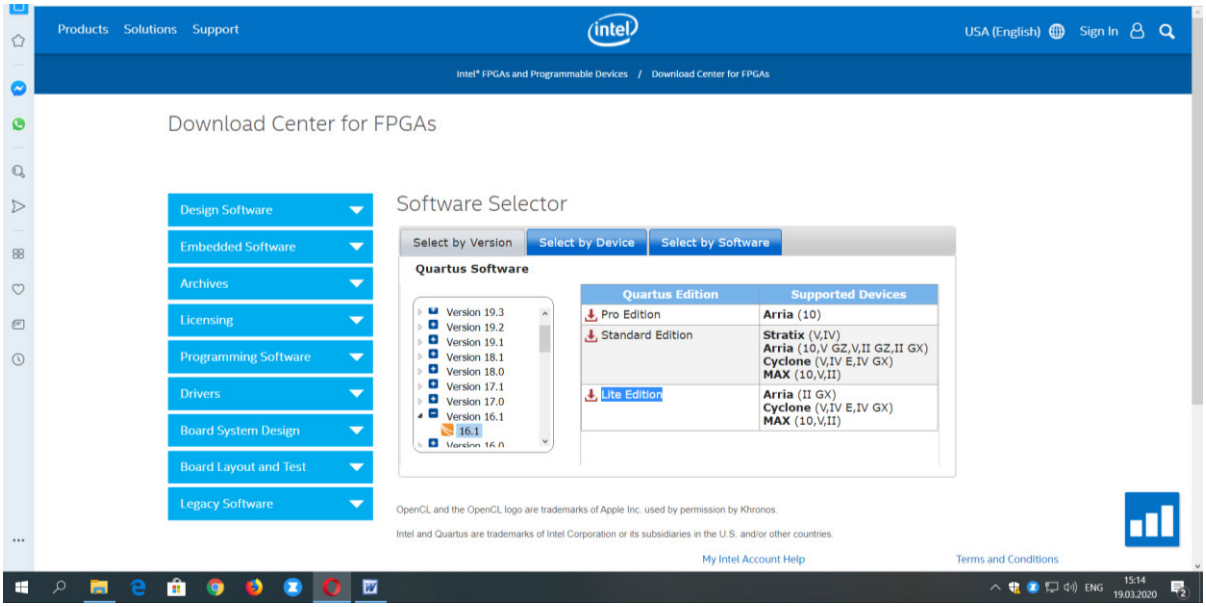
Житомир
2023

ЗМІСТ

ЛАБОРАТОРНА РОБОТА №1
Синтез логічних схем малої інтеграції в ПЛІС в графічному режимі з використанням САПР Altera QUARTUS II

1. Мета роботи

Метою роботи є знайомство з САПР для ПЛІС Altera Quartus II, а також вивчення способів синтезу логічних схем і мінімізації логічних функцій. (https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html) наприклад Version 16.1 – Lite Edition.



В Quartus Prime Lite Edition скачати: Quartus Prime (includes Nios II EDS), ModelSim-Intel FPGA Edition (includes Starter Edition), Cyclone IV device support.

2. Короткі теоретичні відомості

2.1.Способи синтезу логічних схем

Будь-яка логічна схема без пам’яті повністю описується таблицею істинності. Ця таблиця є вихідною інформацією для синтезу схеми на основі логічних елементів "І", "АБО", "НІ". Для розробки потрібного цифрового приладу спочатку на основі таблиці істинності записують його логічний вираз. Потім з метою спрощення цифрового пристрою мінімізують його

логічний вираз і розробляють схему, яка виконує отриманий логічний вираз. Логічний вираз можна отримати двома способами:

- на основі досконалої диз'юктивної нормальної форми;
- на основі досконалої кон'юктивної нормальної форми.

Досконала диз'юктивна нормальна форма (ДДНФ)представлена сумою груп. Кожна група являє собою***мінтерм*** – добуток, в який входять всі змінні.

Наприклад,

$$F(A,B,C) = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C}$$

Досконала кон'юктивна нормальна форма (ДКНФ) представляється добутком груп. Кожна група являє***макстерм*** – сума, в яку входять всі змінні

Наприклад,

$$F(A,B,C) = (\bar{A} + B + C) \cdot (A + \bar{B} + C) \cdot (A + B + \bar{C})$$

Якщо схема має декілька виходів, то кожен вихід описується своєю функцією. Така система функцій називається системою власних функцій.

Приклад:

Таблиця 1.1

Таблиця істинності для функцій трьох змінних

A	B	C	Y	Мінтерм	Макстерм
0	0	0	0	$\overline{ABC}$	$A+B+C$
0	0	1	1	$\bar{A}\bar{B}C$	$A+B+\bar{C}$
0	1	0	0	$\bar{A}B\bar{C}$	$A+\bar{B}+C$
0	1	1	1	$\bar{A}BC$	$A+\bar{B}+\bar{C}$
1	0	0	0	$A\bar{B}\bar{C}$	$\bar{A}+B+C$

1	0	1	0	$A\bar{B}C$	$\bar{A}+B+\bar{C}$
1	1	0	1	$AB\bar{C}$	$\bar{A}+\bar{B}+C$
1	1	1	1	ABC	$\bar{A}+\bar{B}+\bar{C}$

ДДНФ складається на основі таблиці істинності шляхом додавання (логічне "АБО") всіх тих мінтермів, для яких вихід Y має значення 1.

$$Y = \bar{A}\cdot B\cdot C + \bar{A}\cdot B\cdot C + A\cdot B\cdot \bar{C} + A\cdot B\cdot C$$

ДКНФ складається на основі таблиці істинності шляхом добутку (логічне "І") всіх макстермів, для яких вихід Y має значення 0.

$$Y = (A+B+C)\cdot(A+B\cdot C)\cdot(\bar{A}+B+C)\cdot(\bar{A}+B+\bar{C})$$

На основі отриманих виразів можна скласти схему пристрою, що реалізує задану функцію. Схема пристрою, отримана на основі ДДНФ, зображена на рис. 1.1, ф на основі ДКНФ на мал. 1.2.

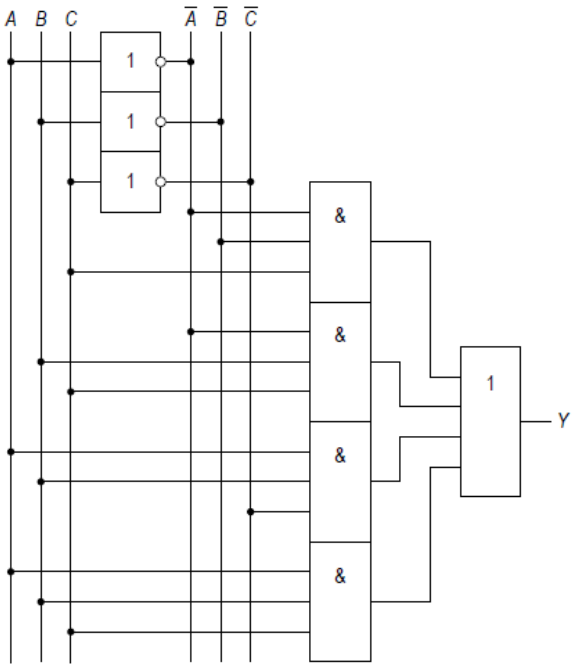


Рис. 1.1 – Схема приладу, отриманого на основі ДДНФ

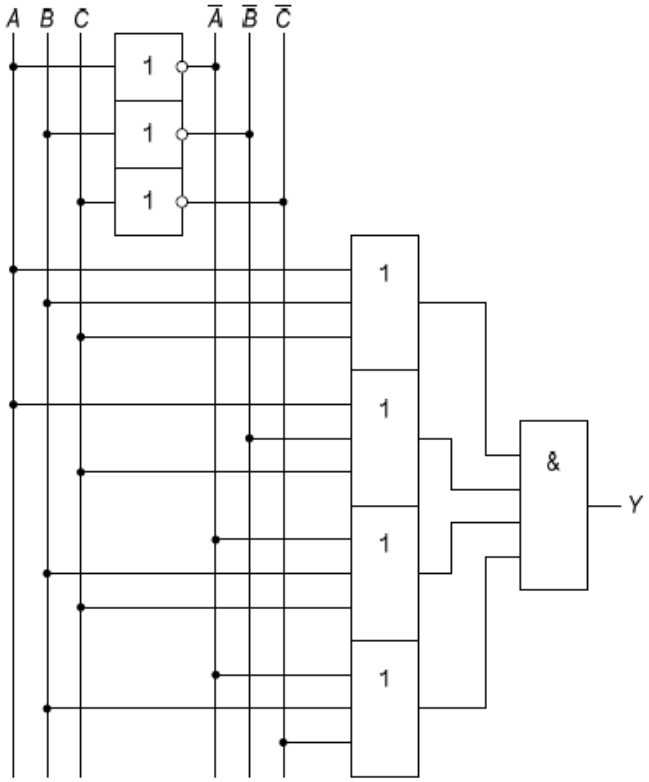


Рис. 1.2 – Схема приладу, отриманого на основі ДКНФ

2.2 Карти Карно

З метою спрощення схеми цифрового приладу використовують мінімізацію функції. ***Карти Карно*** є наглядним методом для спрощення булевих рівнянь.

На рис. 1.3 показана карта Карно для функції, описаній в табл. 1.1. Верхній рядок карти дає 4 можливих значень для змінних *A*і *B*. Ліва колонка дає 2 можливих значень змінної *C*. Кожна клітинка карти Карно відповідає рядку таблиці істинності і містить значення функції*Y* із цього рядка. Наприклад, верхня ліва клітинка відповідає першому рядку таблиці істинності і показує, що значення функції *Y*будедорівнювати0, коли *ABC* = 000. Як і кожний рядок в таблиці істинності, кожна клітинка карти Карно є визначений мінтерм.

Кожна клітинка (мінтерм) відрізняється від сусідньої зміною тільки

однієї змінної. Це означає, що сусідня клітинка відрізняються тільки в значені одного літерала, значення якого "істина" в одній клітинці і "обман" в сусідній. Змінні Аі В комбінуються в верхньому рядку у вигляді ***коду Грея***: 00, 01, 10, 11. На відміну від бітового порядку по зростанню величини (00, 01, 10, 11), в коді Грея сусідні записи відрізняються тільки на один розряд.

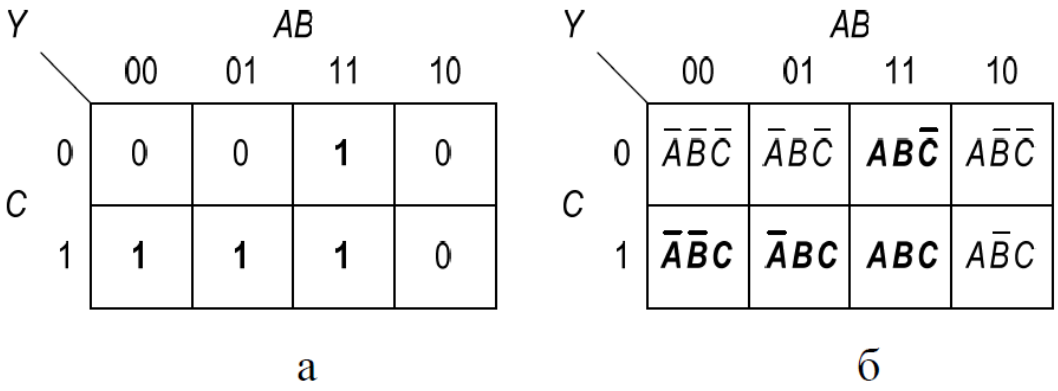


Рис.1.3 – Функція трьох змінних:
а – карта Карно, б – ката Карно з мінтермами

Карти Карно також "кільцева". Клітинка з самого правого краю таблиці є сусідньою із самою лівою, так як вона відрізняється тільки в одній змінній (*А*). Можна згорнути карту в циліндр, з’єднавши краї, і навіть в цьому випадку сусідні клітинки також будуть відрізнятися тільки в одній змінній.

На карті Карно рис. 1.3 містить чотири одиниці, що відповідають числу мінтермів в рівнянні. Читання мінтермів із карт Карно в точності відповідає читанню ДДНФ із таблиці істинності.

Карти Карно допомагають спрощувати графік, обводячи одиниці в сусідніх клітинках овалами, як це показано на рис. 1.4. Для кожного овалу пишуть і відповідну йому імпліканту. Змінні, для яких пряма і комплементарна форми попадають в один овал, виключаються.

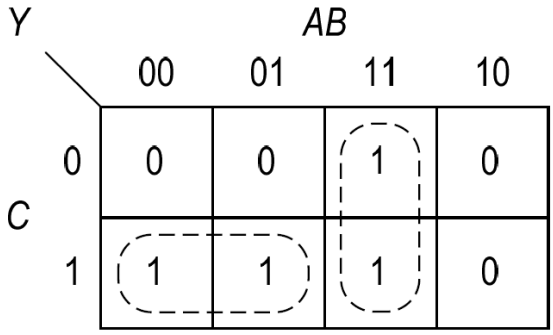


Рис. 1.4 – Мінімізація за допомогою карти Карно

Таким чином, отримуємо функцію $Y = \bar{A}C + AB$, на основі якої створимо нову схему пристрою (рис. 1.5).

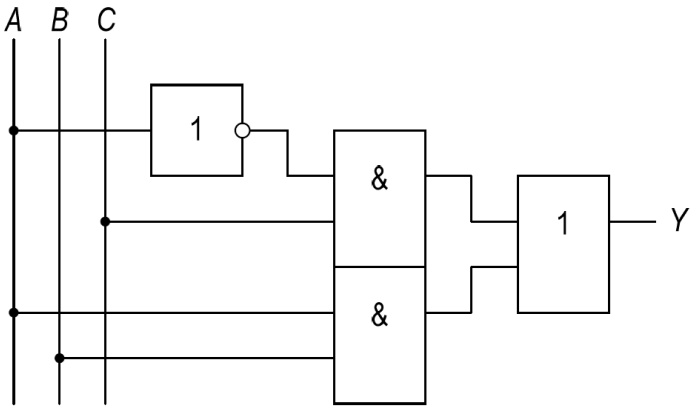


Рис. 1.5 – схема приладу, отриманого після мінімізації логічних функцій

Порівняйте з рис. 1.1 і 1.2, схема вийде на багато компактнішою.

Правила для знаходження мінімального рівняння із карт Карно наступні:

- Використовується менше всього овалів, необхідних для покриття всіх 1;
- Всі клітинки в кожному овалі обов’язково містять 1 або значення X);
- Кожен овал повинен охоплювати блок, число клітинок якого в кожному напрямку рівне степеню двійки (тобто 1, 2, 4, 8 і т.д.);
- Об’єднати клітинки можна тільки по горизонталі чи вертикалі;

- Кожен овал повинен бути на стільки великим, на скільки це можливо;
- Овал може зв’язувати краї карти Карно;
- Одиниця на карті Карно може бути обведена безліч разів, якщо це дозволяє зменшити число овалів, які будуть використовуватися.

3. Завдання до роботи

1. Синтезувати логічну схему на основі дворівневої логіки (ДКНФ або ДДНФ) по таблиці істинності свого варіанту (див табл. 1.3).

2. Запустити в середовище Altera Quartus II і створити новий проект (меню File → New Project Wizard…). Майстер створення проекту містить 5 кроків:

- Крок 1 – ім’я проекту і директорія його зберігання (рис. 1.6).

  Увага! Не допускати використання в іменах файлів проекту і в шляху їх зберігання кириличних символів. Для зберігання кожного проекту використовувати окрему папку на диску.

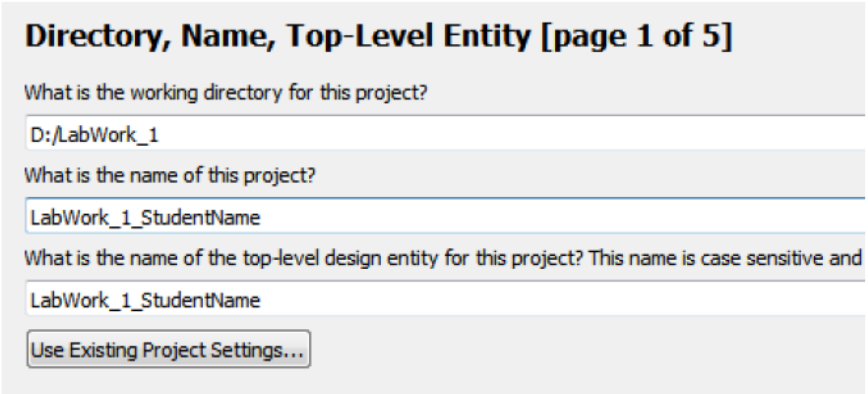


Рис. 1.6 – Вибір директорії й імені при створенні проекту

- Крок 2 – Додавання в проект других необхідних файлів. В даній роботі етап необхідно пропустити.

- Крок 3 – Вибір сімейства і назви ПЛІС, під яку створюється проект. Необхідно вказати сімейство Cyclone IV E, назву EP4CE115F29C7. Щоб прискорити процес вибору типу ПЛІС, можна використати фільтри по корпусу, числу виводів, класу швидкості, а також по імені, ввівши символи, що входять до назви (див. рис. 1.7).

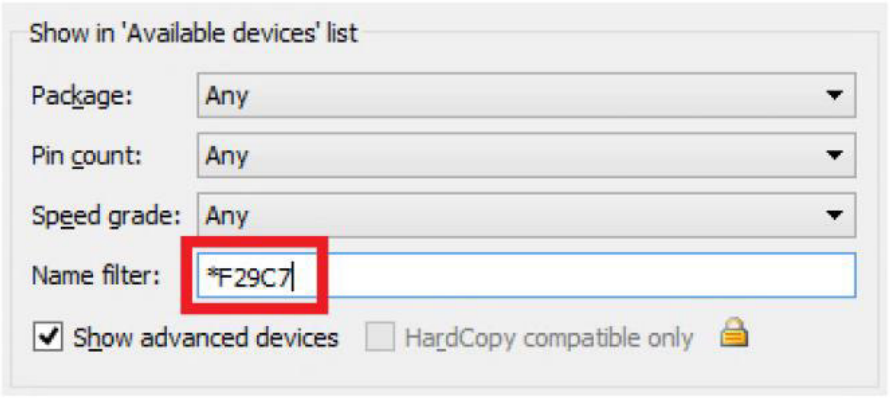


Рис. 1.7 – Використання фільтра по імені при виборі типу ПЛІС

- Крок 4 – Вибір інструментів для синтезу, налагодження і симуляції. В даній роботі етап необхідно пропустити.
- Крок 5 – На останньому кроці відображається сумарна інформація по проекту. Для завершення створення проекту потрібно натиснути кнопку "Finish".

3. Добавити в проект новий файл (File → New…) типу "Block Diagram/Schematic File" і накреслити синтезовану схему, використовуючи інструменти "Orthogonal Node Tool" (з'єднувальні провідники),"Pin Tool" (вхідні і вихідні порти), "Symbol Tool" (логічні елементи). Altera Quartus IIуже містить в своїй бібліотеці стандартні символи простих логічних елементів ("and", "ar4", "not"і т.д.) (див. рис. 1.8).

4. Зберегти файл, не змінюючи назви файлу, і виконати аналіз і синтез проекту (Processing → Start →Start Analysis & Synthesis). У випадку відсутності помилок зробити призначення виводів (меню Assignments → Pin

Planner) на полі Location нижній частині вікна Pin Planner відповідно до першого і другого рядка табл. 1.2.

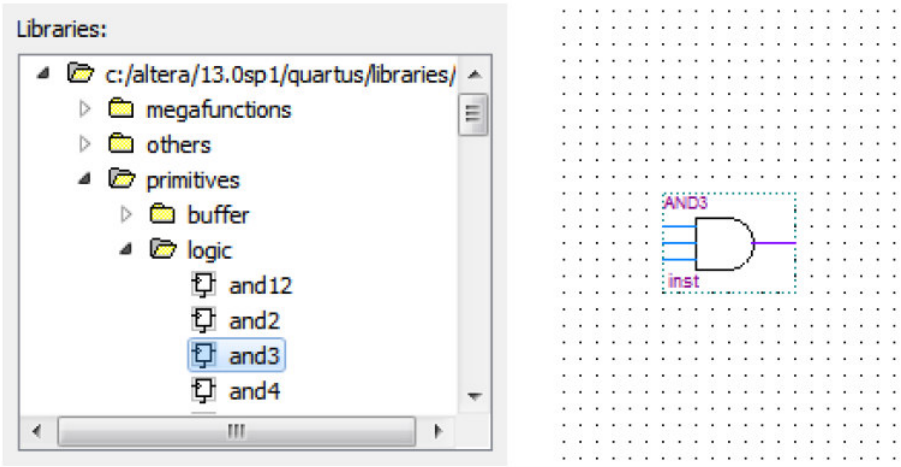


Рис. 1.8 – Бібліотека стандартних логічних елементів в Altera Quartus II

Для того щоб отримати часову діаграму потрібно (https://www.youtube.com/watch?v=a8JAkKhxlQI) вибрати File → New…University Program VWF. Далі натиснути правою кнопкою на білому полі ліворуч і вибрати Insert Node or Bus + Node Finder. Далі List, виділити усі піни і натиснути в центрі на « > ». + Ok, Ok. Далі лівою миші на тискаємо на pin in і в меню вибираємо Edit – Value – Overwrite clock, вибираємо період прямокутного сигналу. Edite – Set End time (us - мікросекунди) – можна виставити тривалість симуляції. В кінці Simulation – Run functional simulation.

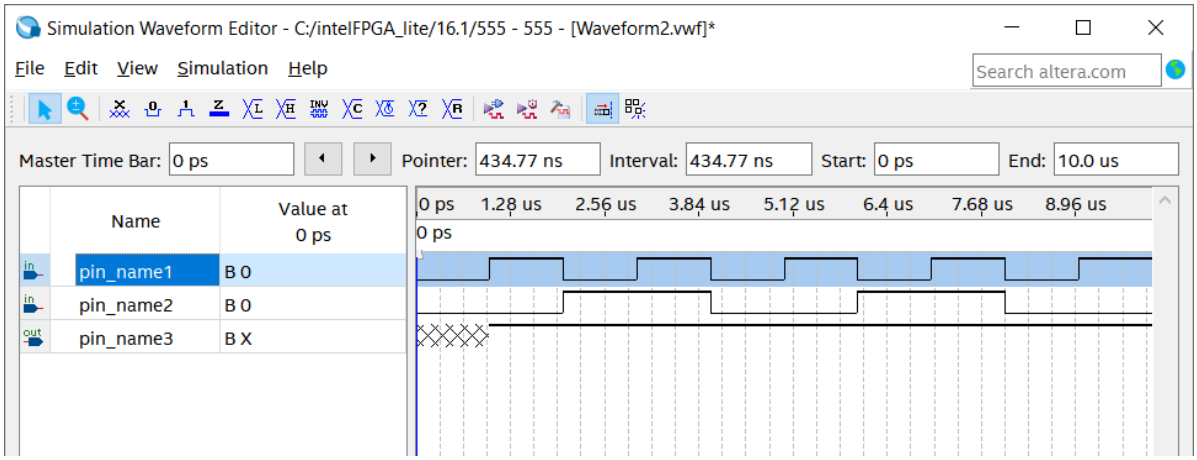


*Рис. 1.9 – Вікно часової діаграми (*University Program VWF)

5. Виконати повну компіляцію проекту (Processing → Start Compilation). В процесі компіляції створюється конфігураційний файл *.sof в папці з проектом в підпапці "output_files". Він використовується при програмуванні ПЛІС.

Таблиця 1.2

Назви виводів схеми

Сигнал	A	B	C	Y
Вивід ПЛІС	AC27	AC28	AB28	G19
На стенді	SW2	SW1	SW0	LEDR0

6. В присутності викладача підключити лабораторний стенд до комп’ютеру і включити на стенді живлення.

7. Запрограмувати навчальний стенд (меню Tools → Programmer) і, встановлюючи на входах схеми за допомогою перемикачівSW2..SW0 всі можливі кодові комбінації із таблиці істинності і нагляду за світло діодам LEDR0, перевірити роботу схеми.

8. Скласти мінімізовану логічну функцію по таблиці істинності свого варіанту (див. табл. 1.3) за допомогою карти Карно. Синтезувати схему по мінімізованій функції.

9. Накреслити нову схему в Altera Quartus II в тому ж файлі і виконати аналіз і синтез проекту. Потім зробити призначення виводів і повну компіляцію проекту.

10. Запрограмувати навчальний стенд і перевірити роботу схеми на відповідність до заданої таблиці істинності.

11. Вимкнути живлення лабораторного стенду і від'єднати його від комп'ютера.

12. Зробити висновки по роботі.

4. Вміст звіту

1. Мета роботи.
2. Задана таблиця істинності і синтезована схема.
3. Карта Карно, синтезована логічна функція і схема.
4. Висновки по роботі.

5. Варіанти завдань

Таблиця 1.3

Варіанти завдань

Варіанти			**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
A	**B**	**C**	**Y**														
0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	0	1
0	0	1	1	0	0	1	0	0	0	1	1	1	0	1	0	1	1
0	1	0	0	1	1	1	1	0	1	0	0	1	0	0	1	0	0
0	1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	1	0
1	0	0	1	0	1	0	1	1	1	0	0	0	1	1	1	0	1
1	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	0	1
1	1	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0
1	1	1	0	0	0	1	1	1	0	0	1	1	0	1	0	1	0

6. Контрольні запитання

1. Що таке ДДНФ/ДКНФ?
2. Як записати ДДНФ/ДКНФ,використати таблицю істинності?
3. Як краще синтезувати логічний пристрій (на основі ДДНФ або ДКНФ), якщо значення функції в таблиці істинності мають більше нулів, чим одиниць?
4. Як розробити логічний пристрій, якщо він має декілька виходів?

5. Що таке мінімізація логічного виразу, і які існують способи мінімізації?

6. Яким чином здійснюється мінімізація функції за допомогою карти Карно?

7. Розкажіть, як визначити таблицю істинності логічного приладу експериментально, використовуючи лабораторний стенд?

8. Написати таблицю істинності для основних логічних вентилів (І, АБО, НІ, виключне АБО).

9. Що таке проект Altera Quartus II? Для чого необхідна компіляція проекту в Altera Quartus II і як її виконати?

10. Для чого проводиться операція призначення виводів в Altera Quartus II?

ЛАБОРАТОРНА РОБОТА №2
Синтез логічних схем перетворювача кодів та дешифратора в ПЛІС мовою Verilog або VHDL з використанням САПР Altera QUARTUS II

1. Мета роботи

Метою роботи являється вивчення принципів дії комбінаційних схем: дешифратора, коду для семисегментного індикатора, суматора, а також знайомство з мовами проектування апаратури.

2. Короткі теоретичні відомості

2.1. Мови проектування апаратури

VHDL* – мова опису апаратури інтегрованих схем. VHDL є базовою мовою при розробці апаратури сучасних обчислюваних систем. Був розроблений в 1983 р. по замовленню Міністерства оборони США з метою формального опису логічних схем для всіх етапів розробки електронних систем, починаючи з модулів мікросхем і закінчуючи великими обчислювальними системами.

Програма на VHDL складається із одного чи декількох файлів. В одному файлі розміщується одна чи декілька пар елементів "об’єкт проекту – архітектура об’єкту". *Об’єкт проекту* – це частина інтерфейсу, в якій вказана інформація про те, як ввімкнути даний об’єкт в середині іншого об’єкту, який знаходиться на більш високому рівні ієрархії. А в *архітектурі об’єкту* описується алгоритм його функціонування. У одного об’єкт може містити декілька архітектур, відповідаючи різним алгоритмам функціонування, проектам на різних етапах проектування.

В прикладах цього і подальшого коду *курсивним шрифтом* вказані коментарі, **напівжирним** – ключові слова даної мови.

Об’єкти проекту виглядають наступним чином:

Опис архітектури:

```
Library std;                          --підключення бібліотеки STD
use std.standard.all;                 -- використовуються всі типи пакета
                                          STANDART бібліотеки STD
entity FUNC_NAND is                   -- назва об'єкту - FUNC_NAND
    port (                            -- оголошення портів
        A : in BIT;                   -- вхідний порт A типуBIT
        B : in BIT;                   -- вхідний порт B типуBIT
        C : out BIT);                 -- вихідний порт C типуBIT
End FUNC_NAND;                        -- кінець оголошення об'єкту
```

Ключове слово *port* відкриває опис входів-виходів (портів) об'єкти. Слово *in*вказує на вхід, а *out*– на вихід. BIT– це тип порту, який, згідно визначенню цього типу в пакеті *standard*, приймають значення 0 і 1.

Опис архітектури:

```
Architecture BEHAVIORAL of FUNC_NAND -- початок архітектури
is                                       BEHAVIORAL об'єкту
                                         FUNC_NAND
-- початок описової (__) частини архітектури
Begin
    C <= not (A and B);
                                         -- паралельний оператор
End BEHAVIORAL;
                                         -- кінець архітектури
```

Розділ архітектури складається із декларативної і описової частин. В декларативний частині оголошуються використані в середині сигналу, константи спеціальні (власні) типи, атрибути, процедури і функції. Описова частина складається із списку паралельних операторів. Всі паралельні оператори виконуються одночасно, їх порядок в списку не має значення.

Verilog*, Verilog HDL – ще одна мова описання архітектури, що використовується для описання і моделювання електронних систем. Verilog була створена в 1984 році фірмою Automated Integrated Design Systems (пізніше перейменований на GatewayDesignAutomation) як власний засіб моделювання. Після поглинання останньої фірмової Caddence, мова почала отримувати все більший попит серед розробників і стала не менш популярною, чим VHDL.

Синтаксис Verilog дуже схожий га синтаксис мови С, що спрощує його освоєння. Препроцесор Verilog дуже схожий на препроцесор мови С, і основні управляючі конструкції *if, while* та інші подібні однойменні конструкції мови С. Програма на Verilog зазвичай складається із декількох модулів. Синтаксис оголошення модуля виглядає наступним чином:

```
Module FUNC_NAND (A, B, C);        // Вхідні пори
        Input A, B;
                                   // Вихідні порти
        output C;
                                   // операція і-ні
        assign C = ~ (A & B);
end module
```

У заголовку модуля описується список портів. У тілі модуля описується його функціональність.

У проекті, особливо в складному, буває багато модулів, з’єднаних між собою. Перш за все, потрібно помітить, що зазвичай в проекті завжди є один *модуль самого верхнього рівня (top level).* Він складається із декількох інших модулів. Ті в свою чергу можуть містити ще модулі і так далі. Необов’язково, щоб всі модулі були написані на одній мові описання апаратури. Зовсім навпаки. Досить зручно і наочно мати модуль самого верхнього рівня виконаним у вигляді схеми, що складається із модулів більш низького рівня.

Ці модулі можуть бути написані різними розробниками,на різних мовах (Verilog, VHDL) і навіть виконанні у вигляді схеми.

System Verilog являє собою надмножину мови Verilog стандарту 2005 року, але з великими можливостями для верифікації і моделювання розробок.

Примітка – при називанні програмних об'єктів (проектів, інтерфейсів, архітектур, пакетів, сигналів, змінних, констант, компонентів і т.д.) в коді потрібно притримуватись правил, наведених в Програмі А даного посібника.

2.2. Дешифратор (декодер)

В спільному випадку *дешифратор* (декодер) – це пристрій, що перетворює цифровий сигнал в якому-небудь кодуванні в іншу, не закодовану форму. *Класичний дешифратор* представляє собою пристрій для перетворення N-розрядного позиційного двійкового коду в одиничний вихідний сигнал на одному із 2^N виходів. При кожній вихідній комбінації сигналів на одному із виходів з'являється логічна 1. Таким чином, по одиничному сигналів на одному із виходів можна судити про вхідній кодовій комбінації. Таблиця істинності класичного дешифратора з двома входами зображена в табл. 2.1.

Табл. 2.1

Таблиця істинності двохрозрядного дешифратора

X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Для побудови схеми дешифратора по таблиці істинності використаємо методику з лабораторної роботи №1. Наприклад, пристрій повинен мати

чотири виходи. Для кожного виходу записуємо логічний вираз. На основі ДДНФ:

$$Y0 = \bar{X1} \cdot \bar{X0}, \qquad Y1 = \bar{X1} \cdot X0, \qquad Y2 = X1 \cdot \bar{X0}, \qquad Y3 = X1 \cdot X0$$

За цією системою виразів не складно побудувати схему потрібного дешифратора (рис. 2.1).

2.3. Перетворювач коду для семисегментного індикатора

Найбільш широкими перетворювачами кодів відомі у використанні до цифрових індикаторів. Наприклад, часто потребуються перетворювачі позиційного двійкового коду в десяткові цифри.

За допомогою семисегментного індикатора (рис. 2.2а) можна висвітити десяти арабських цифр (рис. 2.3). Такі індикатори виготовляються в двох варіантах – зі спільним анодом (ОА, рис. 2.2б) і зі спільним катодом (ОК, рис. 2.2в). Слід помітити, що "запалювання" сегменту індикатора з ОА виконується логічним нулем, ф з ОК – логічною одиницею.

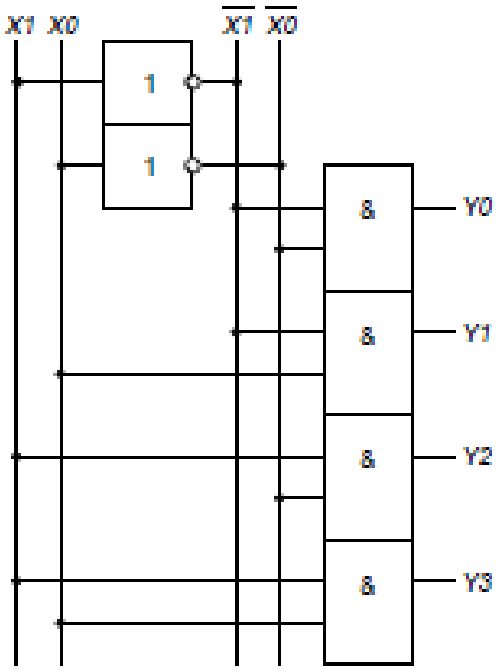


Рис. 2.1 – схема дешифратора

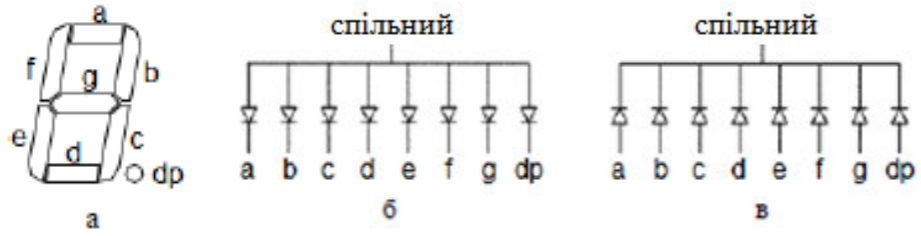


Рис. 2.2 – Семисегментний індикатор з точкою:

а – зовнішній вигляд, б – схема зі спільним анодом, в – схема зі спільним катодом

Рис. 2.3 – Приклад відображення цифр на семисегментному індикаторі

Очевидно, що двійковий код повинен мати не менше 4-х розрядів (2^4 = 16, що більше 10). Умовне графічне позначення представлене на рис. 2.4.

Таблиця 2.2

Таблиця істинності перетворювача колу

Цифра	D3	D2	D1	D0	Ha	Hb	Hc	Hd	He	Hf	Hg
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

-	Всі інші комбінації	0	0	0	0	0	0	1

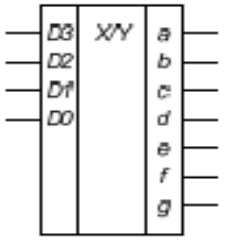


Рис.2.4 –Умовне графічне позначення перетворювача коду

2.4. Суматор

Напівсуматор має два входи (*A* і *B*) і виходи (*S* і $C_{вих.}$). S – це сума *A* і *B*. Якщо ж і *A*, і *B* дорівнюють 1, тоді вихід *S* повинен дорівнювати 2, але таке число не може бути представлене у вигляді одного двійкового розряду. В цьому випадку результат вказується разом з переносом $C_{вих.}$ в наступний рядок. Напівсуматор може бути збудований із елементів XOR ("виключне АБО") і AND ("логічне І"), рис. 2.5.

Таблиця 2.3

Таблиця істинності напівсуматора

A	*B*	$C_{вих.}$	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B \qquad C_{вых} = AB$$

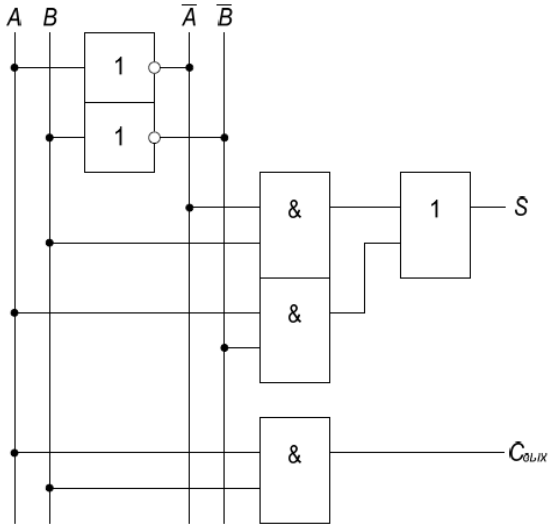


Рис. 2.5 – Схема напівсуматора

Повний суматор окрім виходу переносу має вхід переносу з попереднього розряду складань $C_{вх.}$.

$$S = A \oplus B \oplus C_{вх}$$

N-розрядний суматор складає два *N*-розрядних числа (*A*і*B*), а також вхідний переніс $C_{вх.}$ І формує *N*-розрядний результат *S* і вихідний переніс $C_{вих.}$. Такий суматор називається ***суматором із переносом, що розповсюджується,*** так як вихідний перенос одного розряду переходить в наступний рядок.

Таблиця 2.4

Таблиця істинності повного суматора

$C_{вх.}$	A	B	**$C_{вих.}$**	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1

0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

3. Завдання до роботи

1. Запустити середовище Altera Quartus II і створити новий проект параметрами:

- Назва проекту: LabWork_2_StudentName
- Сімейство ПЛІС: Cyclone IV E
- Тип ПЛІС:EP4CE115F29C7

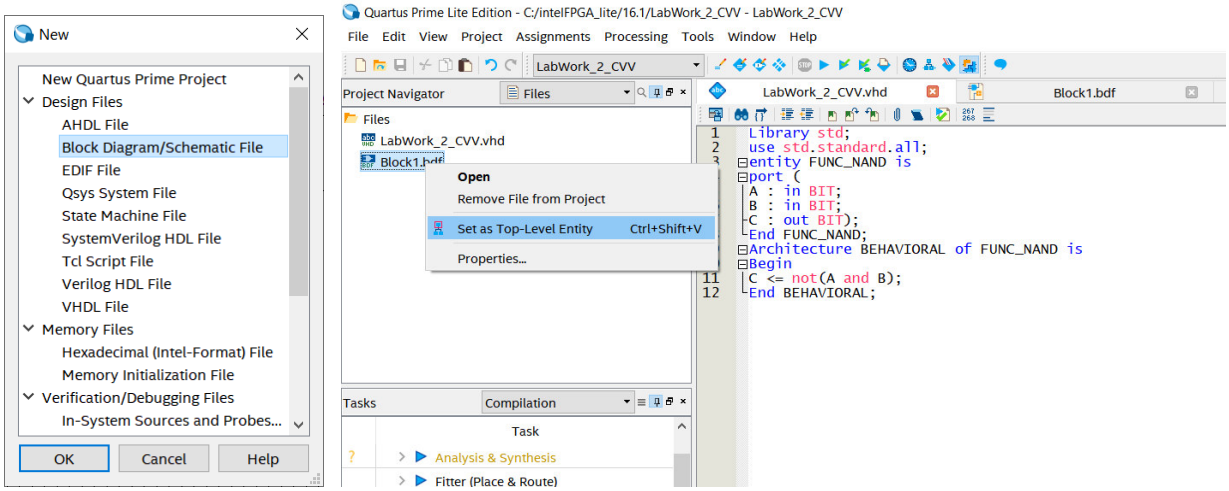
Інше залишити по замовчуванню.

2. Добавити в проект три нові файли типу "VHDL File" (або "Verilog HDL File" якщо пишеться програма мовою Verilog) і описати схеми дешифратора 2×4, перетворювача коду для семисегментного індикатора і повного суматора.

Для цього в головному меню натискаємо: File – New – VHDL File + Ok. У відкритому вікні набираємо текст програми мовою VHDL.

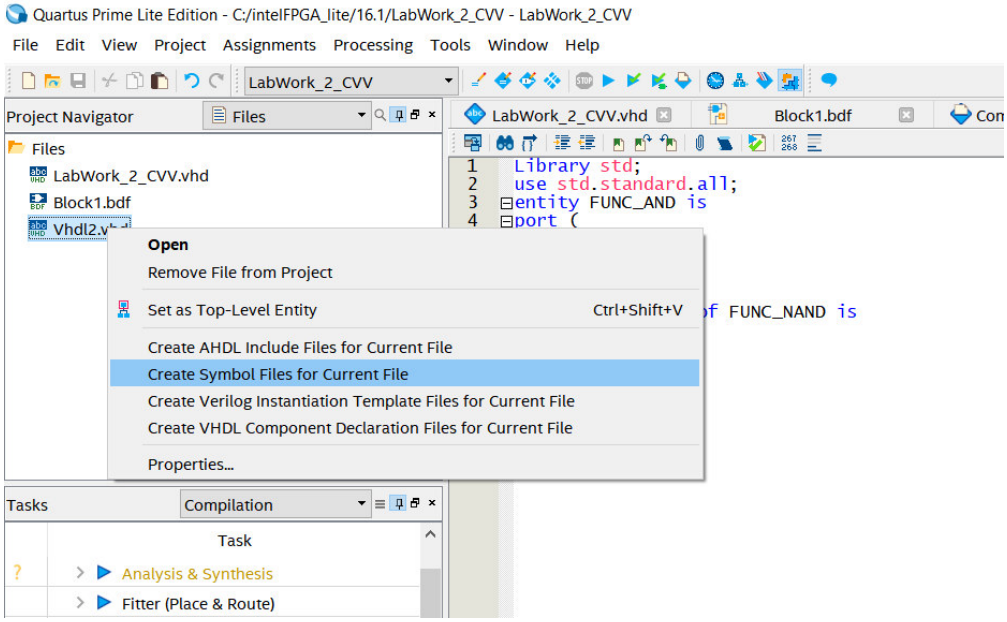
Увага! Необхідно врахувати, що на стенді семисегментні індикатори підключені по схемі зі *спільним катодом.*

3. Додати до проекту файл типу "Block Diagram/Schematic File" і назначити його файлом верхнього рівня (Project → Set as Top-Level Entity):

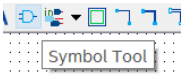


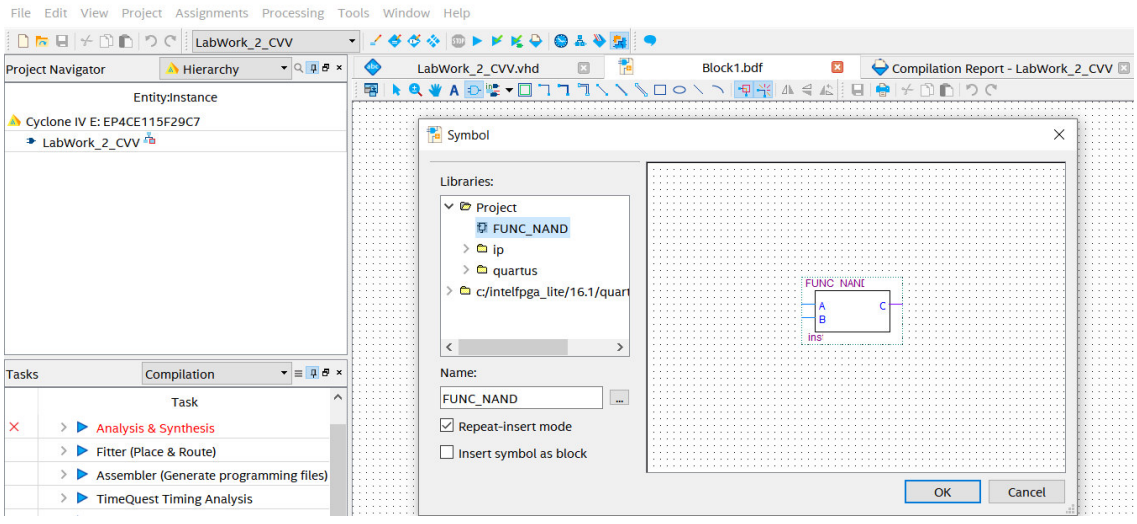
4. Виконати аналіз і синтез проекту.

5. Виконати генерацію схемного синтезу символу із HDL-опису для кожного модуля окремо(File → Create/Update → Create Symbol Files from Current File):



Після цієї операції символи стануть доступні в бібліотеці інструментів "Symbol Tool" в підпапці "Project":





6. В файлі схеми, створеному в п. 3, накреслити спільну схему, що містить в собі всі досліджувані модулі.

7. Виконати аналіз і синтез проекту і призначити виводи у відповідності до таблиць 2.5–2.7. Після цього виконати повну компіляцію проекту.

Таблиця 2.5

Призначення виводів дешифратора 2×4

Сигнал	**X1**	**X0**	**Y3**	**Y2**	**Y1**	**Y0**
Вивід ПЛІС	AC27	AC28	F21	E19	F19	G19
На стенді	SW2	SW1	LEDR3	LEDR2	LEDR1	LEDR0

Таблиця 2.6

Призначення виводів перетворювача коду для семисегментного індикатора

Сигнал	**D3**	**D2**	**D1**	**D0**
Вивід ПЛІС	Y23	Y24	AA22	AA23
На стенді	SW17	SW16	SW15	SW14

Сигнал	**Ha**	**Hb**	**Hc**	**Hd**	**He**	**Hf**	**Hg**
Вивід ПЛІС	AD17	AE17	AG17	AH17	AF17	AG18	AA14
На стенді	HEX7						

Таблиця 2.7

Призначення виводів повного суматора

Сигнал	**$C_{вх.}$**	**A**	**B**	**$C_{вих.}$**	**S**
Вивід ПЛІС	AC26	AB27	AD27	E18	F18
На стенді	SW5	SW4	SW3	LED5	LED4

8. Підключити до комп’ютера лабораторний стенд, ввімкнути його і запрограмувати. Подаючи різні вхідні сигнали і спостерігати за станом індикаторів, проаналізувати роботу досліджуваної схеми.

4. Вміст звіту

1. Мета роботи.

2. Схеми дослідження дешифратора, перетворювача коду для семисегментного індикатора, суматора.

3. Вихідні коди модулів дешифратора, перетворювача коду для семисегментного індикатора, суматора.

4. Таблиця істинності для кожної схеми.

5. Висновки по роботі.

5. Контрольні запитання

1. Поясніть принцип роботи дешифратора і шифратора.

2. Як синтезувати дешифратор із заданою розрядністю?

3. Як працює перетворювач коду для семисегментного індикатора? Як влаштований семисегментний індикатор?

4. Як працює мультиплікатор і демультиплікатор?

5. Як працює повний суматор? Чим він відрізняється від напівсуматора?

6. Що таке біт переносу в суматорі?

7. Що таке файл верхнього рівня в Altera Quartus II?

8. Опишіть структуру VHDL-файлу (Verilog-файлу).

Лабораторна робота №3
Дослідження тригерів та регістрів в ПЛІС в графічному режимі з використанням САПР Altera QUARTUS II

1. Мета роботи

Метою роботи є експериментальне дослідження роботи різних типів тригерів і регістрів.

2. Короткі теоретичні відомості

2.1. Тригери

Тригери – прості послідовних схема, що запам'ятовують один біт інформації. Використання тригерів дозволяє реалізовувати пристрої оперативної пам'яті (тобто пам'яті, інформація в якій зберігається тільки на час обчислень). Однак тригери можуть використовуватися і для побудови деяких цифрових пристроїв з пам'яттю, таких як лічильники, перетворювачі послідовного коду в паралельний або цифрові лінії затримки.

Однією з найпростіших послідовних схем є RS-тригер, що складається, як показано на рис. 3.1, з двох перехресно включених елементів АБО-НЕ. У тригера є два входи – R і S і два виходи Q і $\overline{Q}$.

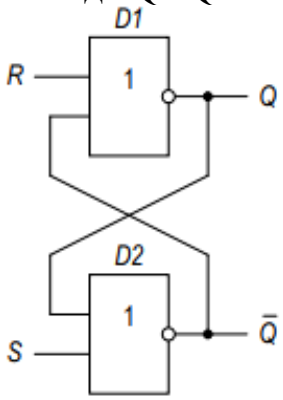


Рисунок 3.1 – Схема RS-тригера

Таблиця переходів, наведена нижче, ілюструє чотири можливих стану тригера. Входи R і S відповідають за скидання (reset) і установку (set) значення на виході Q відповідно.

Таблиця 3.1

Таблиця переходів RS-тригера

S	R	Q	$\overline{Q}$	Режим
0	0	$Q_{пред}$	$\overline{Q_{пред}}$	Режим зберігання
0	1	0	1	Режим установки 0
1	0	1	0	Режим установки 1
1	1	*	*	Заборонений режим

Заборонений режим не використовується, так як стану виходів тригера в загальному випадку не визначені і будуть залежати від його конкретної реалізації.

Розглянутий вище тригер є асинхронним.

У синхронного RS-тригера крім R і S входів є вхід синхронізації). Його схема побудови на основі асинхронного RS-тригера і умовне позначення показані на рис. 3.2.

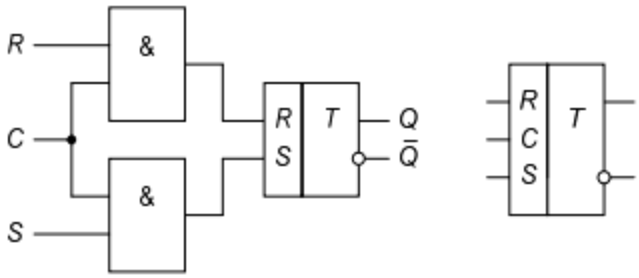


Рисунок 3.2 – Синхроний RS-тригер, а – схема, б – УГЗ

Стан виходів тактованих RS-тригера може змінюватися тільки в моменти приходу тактових імпульсів. В іншому випадку елементи І «замкнені» і не пропускають сигнали R і S. У цьому випадку говорять, що схема працює синхронно.

RS-тригер незручний через незвичайного поведінки в забороненому режимі. D-тригер (D-засувка, D-latch) (рис. 3.3) вирішує цю проблему. Він побудований на основі синхронного RS-тригера і має вхід даних D, що визначає, яким буде наступний стан, і вхід тактового сигналу С.

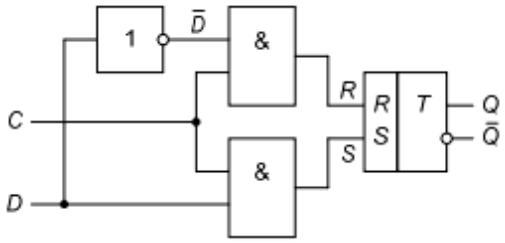


Рисунок 3.3 – Схема D-тригера

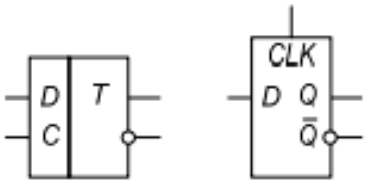
Таблиця 3.2

Таблиця істинності D-тригера

C	D	Q	$\overline{Q}$	Режим
0	X	$Q_{пред}$	$\overline{Q_{пред}}$	Режим зберігання
1	0	0	1	"Прозорий режим"
	1	1	0	

Коли С = 1, говорять, що D-тригер «прозорий», тобто він пропускає дані

D на вихід Q, як якщо б він був звичайним буфером. Коли C = 0 - «непрозорий», тобто не пропускає нові дані з входу D на вихід Q, а Q зберігає своє попереднє значення. Умовне позначення D-тригера представлено на рис. 3.4

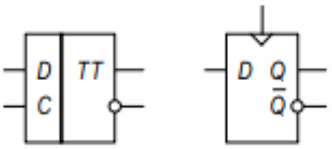


а б

Рисунок 3.4 – Умовне позначення D-тригера
а – вітчизняне, б – закордонне

Стан D-тригера змінюється безперервно, поки C = 1.

Такий триггер називається статичним. D-тригер, синхронізований фронтом (динамічний), може бути побудований з двох включених послідовно статичних D-тригерів, у яких тактові сигнали є булеві доповненнями один одного. Умовне позначення такого D-тригера наведено на рис. 3.5.



а б

Рисунок 3.5 – Умовне позначення D-тригера, який потрібно синхронізувати фронтом: а – вітчизняне, б – закордонне

Динамічний D-тригеркопіюєзначення з D на Q по задньому або передньому (залежить від реалізації тригера) фронту тактового імпульсу і пам'ятаєцей стан весь інший час. Таким чином вхід D визначає нове, майбутнє стан тригера, фронт визначає конкретний момент часу, коли стан буде оновлено.

Робота динамічного D-тригера, щопрацює по задньому фронту синхроімпульсів, представлена в таблиці 3.3.

Таблиця 3.3

Таблиця переходів динамічного D-тригера

C	D	Q	$\overline{Q}$	Режим
0	X	$Q_{пред}$	$\overline{Q_{пред}}$	Режим зберігання
1				
⎍				
⎍	0	0	1	Переключення
⎍	1	1	0	

Т-тригер - це рахунковий тригер. У даного тригера є тільки один вхід Т. Після надходження на цей вхід імпульсу, стан тригера змінюється на протилежне. Рахунковим він називається тому, що Т-тригер як би підраховує кількість імпульсів, що надійшли на його вхід. При отриманні другого імпульсу Т-тригер здатний відобразити тільки молодший розряд числа $2_{10} = 10_2$, тому на його виході знову виникає логічний нуль.

Т-тригер можна синтезувати з будь-якого двоступеневого тригера. Розглянемо приклад синтезу Т-тригера з динамічного D-тригера. Для того щоб перетворити D-тригер в рахунковий, необхідно ввести ланцюг зворотного зв'язку з інверсного виходу цього тригера на вхід, як показано на рис. 3.6.

Таблиця 3.4

Таблиця переходів Т-тригера

T	Q	$\overline{Q}$	Режим
0	$Q_{пред}$	$\overline{Q_{пред}}$	Зберігання
1			
⎍			
⎍	0	1	Переключення
	1	0	

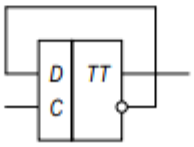


Рисунок 3.6 – Схема Т-тригера на основі двоступеневого D-тригера

Існує ще одне подання Т-тригера. при розробці схем синхронних двійкових лічильників важливо здійснювати одночасний запис у всі його тригери. У цьому випадку вхід Т тригера служить тільки для дозволу зміни стану на протилежне, а синхронізація проводиться окремим входом С. Подібна схема Т-тригера приведена на рис. 3.7.

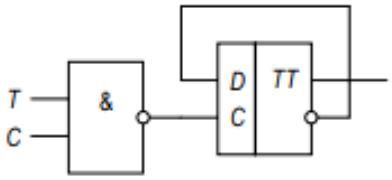


Рисунок 3.7 – Схема синхронного Т-тригера

2.2. Регістри

Регістри призначені для збереження і перетворення багаторозрядних двійкових чисел. ***N-розрядний регістр*** - набір з N тригерів із загальним тактовим сигналом. Таким чином, всі біти регістра оновлюються одночасно. Регістр є ключовим блоком при побудові більшості послідовних схем. На рис. 3.8 показана схема і позначення чотирьохрозрядного регістра зі входами D3..D0 і виходами Q3..Q0.

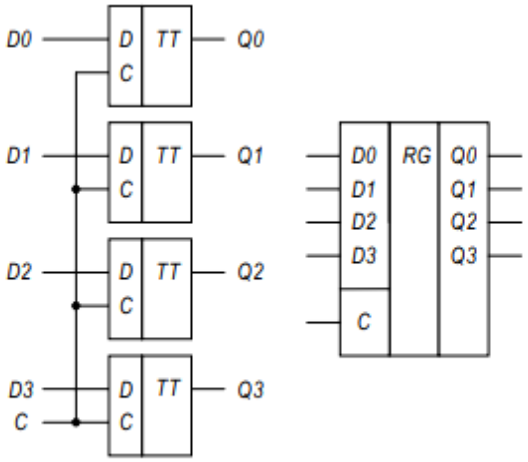


Рисунок 3.8 - Чотирьохрозрядний регістр:
а - схема, б - умовне позначення

Зсувний регістр має вхід тактового сигналу, послідовний вхід Sin, послідовний вихід Sout і N паралельних виходів. По кожному задньому фронту тактового імпульсу в перший тригер регістра записується новий біт зі входу Sin а вміст наступних тригерів зсувається вперед.

Зсувний регістр може бути побудований з N послідовно з'єднаних тригерів (рис. 3.9). Такий реєстр можна розглядати як послідовно-паралельний перетворювач.

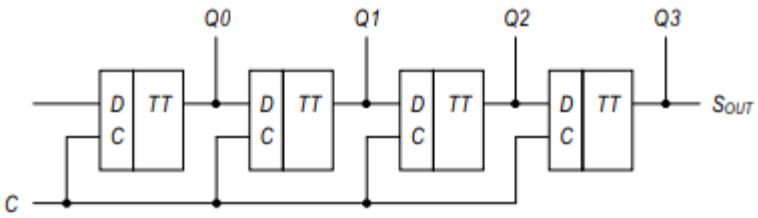


Рисунок 3.9 – Схема 4-розрядного зсувного регістру

В паралельно-послідовний перетворювач паралельно завантажується N біт, які потім послідовно (по одному біти за раз) надходять на вихід.

3. Завдання до роботи

Запустити середовище Altera Quartus II і створити новий проект з параметрами:

- Ім'я проекту: LabWork_3_StudentName
- Сімейство ПЛІС: Cyclone IV E
- Тип ПЛІС: EP4CE115F29C7

Решту залишити за замовчуванням.

3.1. Дослідження RS-тригерів

1. Додати в проект новий файл схеми і накреслити схему асинхронного RS-тригера.

2. Виконати аналіз і синтез проекту і зробити призначення виводів відповідно до таблиці 3.5. Після цього виконати повну компіляцію проекту.

Таблиця 3.5

Призначення виводів асинхронного RS-тригера

Сигнал	R	S	Q
Вивід ПЛІС	AC27	AC28	G19
На стенді	SW2	SW1	LEDR0

3. Запрограмувати навчальний стенд і, спостерігаючи за станами світлодіодних індикаторів, проаналізувати роботу досліджуваної схеми.

4. Перетворити схему в синхронний RS-тригер, виконати аналіз і синтез проекту і зробити призначення виводів відповідно з таблицею 3.6. Після цього виконати повну компіляцію проекту.

Вхідний сигнал синхронізації виробляти з кнопки KEY3. При її натисканні надходить сигнал низького логічного рівня, а при відпуску - високого, тому в ланцюг сигналу синхронізації необхідно ввести інвертор.

Таблиця 3.6

Призначення виводів синхронного RS-тригера

Сигнал	C	R	S	Q
Вивід ПЛІС	R24	AC27	AC28	G19
На стенді	KEY3	SW2	SW1	LEDR0

5. Запрограмувати навчальний стенд і, спостерігаючи за станами світлодіодних індикаторів, проаналізувати роботу досліджуваної схеми.

3.2. Дослідження D-тригера

1. Накреслити схему динамічного D-тригера на основі стандартного

бібліотечного елемента «DFF» (рис. 3.10).

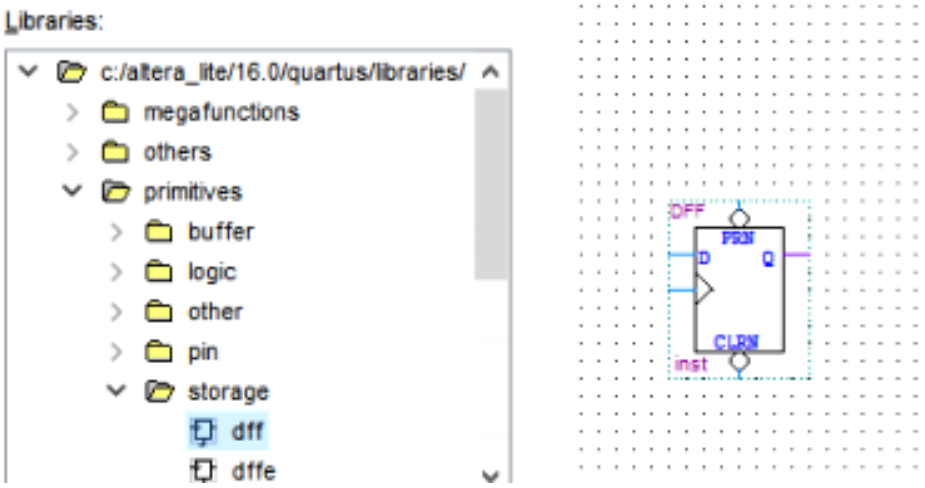


Рисунок 3.10 – D-тригер в бібліотеці стандартних логічних елементів

2. Виконати аналіз і синтез проекту і зробити призначення виводів відповідно до таблиці 3.7. Після цього виконати повну компіляцію проекту.

Таблиця 3.7

Призначення виводів динамічного D-тригера

Сигнал	D	C	Q
Вивід ПЛІС	AC27	R24	G19
На стенді	SW2	KEY3	LEDR0

3. Запрограмувати навчальний стенд і, спостерігаючи за станами світлодіодних індикаторів, проаналізувати роботу досліджуваної схеми.

3.3. Дослідження Т-тригера

1. Накреслити схему Т-тригера на основі стандартного бібліотечного елемента «TFF».

2. Виконати аналіз і синтез проекту і зробити призначення виводів відповідно до таблиці 3.8. Після цього виконати повну компіляцію проекту.

Увага! При використанні введення інформації з кнопок лабораторного стенду виникає брязкіт контактів. Опис цього явища і спосіб боротьби з ним - модуль «button_debouncer» - наведені в Додатку Б. Для того щоб використовувати модуль «Button_debouncer», необхідно перенести код, наведений в Додатку Б, в файл «debouncer.v», а потім включити його в проект. Для цього слід викликати меню «Assigment → Settings» і на вкладціFiles додати файл з модулем - необхідно вказати шлях до файлу, натиснути кнопку «Add», потім «Apply» і «OK».

Таблиця 3.8

Призначення виводів Т-тригера

Сигнал	G_50MHz	T	C	Q
Вивід ПЛІС	Y2	AC27	R24	G19
На стенді	50 MHz Oscillator	SW2	KEY3	LEDR0

3. Запрограмувати навчальний стенд і, спостерігаючи за станами світлодіодних індикаторів, проаналізувати роботу досліджуваної схеми.

3.4. Дослідження паралельного регістра

1. Накреслити схему паралельного 4-розрядного регістра (Див. рис. 3.8).

2. Виконати аналіз і синтез проекту і зробити призначення виводів відповідно до таблиці 3.9. Після цього виконати повну компіляцію проекту.

Таблиця 3.9

Призначення виводів паралельного регістра

Сигнал	G_50MHz	C	D3	D2	D1	D0
Вивід ПЛІС	Y2	R24	AD27	AC27	AC28	AB28
На стенді	50 MHz Oscillator	KEY3	SW3	SW2	SW1	SW0

Сигнал	Q3	Q2	Q1	Q0
Вивід ПЛІС	F21	E19	F19	G19
На стенді	LEDR3	LEDR2	LEDR1	LEDR0

3. Запрограмувати навчальний стенд і по черзі зробити запис декількох 4-розрядних двійкових чисел в регістр.

Спостерігаючи за станами світлодіодних індикаторів, проаналізувати роботу досліджуваної схеми.

3.5. Дослідження регістру зсуву

1. Накреслити схему зсувного 4-розрядного регістра (див. рис. 3.9).

2. Виконати аналіз і синтез проекту і зробити призначення виводів відповідно до таблиці 3.10. Після цього виконати повну компіляцію проекту.

Таблиця 3.10

Призначення виводів паралельного регістра

Сигнал	G_50MHz	C	S_{in}
Вивід ПЛІС	Y2	R24	AB28
На стенді	50 MHz Oscillator	KEY3	SW0

Сигнал	Q3	Q2	Q1	Q0
Вивід ПЛІС	F21	E19	F19	G19
На стенді	LEDR3	LEDR2	LEDR1	LEDR0

3. Запрограмувати навчальний стенд і, спостерігаючи за станами світлодіодних індикаторів, проаналізувати роботу досліджуваної схеми.

4. Зміст звіту

1. Мета роботи.

2. Схеми дослідження тригерів і регістрів і їх умовно-графічне позначення.

3. Таблиці переходів схем тригерів.

4. Висновки по роботі.

5. Контрольні питання

1. Накреслити схему RS-тригера на логічних елементах АБО-НЕ і пояснити принцип його роботи.

2. Чим синхронний RS-тригер відрізняється від асинхронного? Як він реалізується?

3. Пояснити по таблиці переходів роботу D-тригера.

4. Якою характерною особливістю володіє періодична послідовність імпульсів на вході Т-тригера?

5. Якою перевагою володіє динамічний D-тригер?

6. Чим визначається розрядність регістрів?

7. Паралельний регістр: його схема і принцип роботи.

8. Послідовно-паралельний регістр: його схема і принцип роботи.

Лабораторна робота №4
Дослідження та синтез лічильників в ПЛІС в графічному режимі з використанням САПР Altera QUARTUS II

1. Мета роботи

Метою роботи є вивчення універсального двійкового лічильника і придбання навичок у побудові й експериментальному дослідженні лічильників.

2. Короткі теоретичні відомості

Лічильник - пристрій для підрахунку числа вхідних імпульсів. Вони використовуються для побудови схем таймерів або для вибірки інструкцій з ПЗУ в мікропроцесорах. Вони можуть використовуватися як подільники частоти в керованих генераторах частоти (синтезаторах). При використанні в ланцюзі фазового автопідстроювання лічильники можуть бути використані для множення частоти як в синтезаторах, так і в мікропроцесорах.

Основні параметри лічильника:

- M (модуль рахунку) - число стійких станів;
- E (ємність) - максимальне число, яке може бути записано в лічильник (E = M - 1);

2.1. Найпростіший асинхронний двійковий лічильник

Найпростіший асинхронний лічильник являє собою кілька послідовно включених рахункових тригерів (рис. 4.1). Нагадаємо, що за кожним вхідного імпульсу рахунковий тригер змінює свій стан на протилежне.

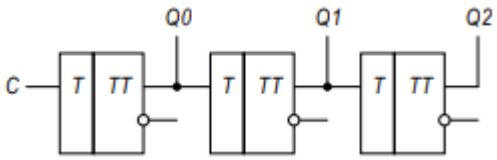


Рисунок 4.1 – Найпростіший сумуючий асинхронний лічильник

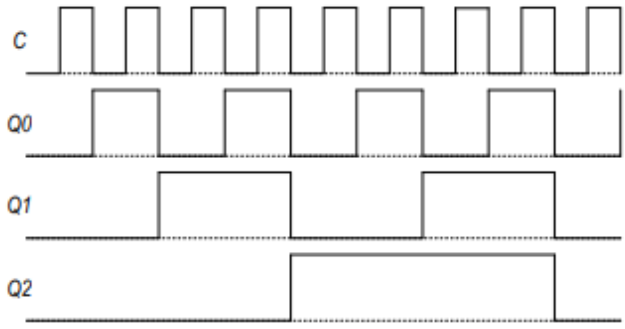


Рисунок 4.2 – Часова діаграма роботи підсумовуючого асинхронного лічильника

Для того щоб розібратися, як працює схема двійкового лічильника, скористаємося часовими діаграмами сигналів на вході і виходах цієї схеми, наведеними на рис. 4.2.

Нехай первісний стан всіх тригерів лічильника буде нульовим. Цей стан видно на часових діаграмах. Після надходження на вхід лічильника тактового імпульсу (який сприймається по задньому фронту) перший тригер змінює свій стан на протилежний, тобто одиницю. Так як по приходу першого імпульсу змінився стан першого тригера, то цей тригер містить молодший розряд двійкового числа (одиниці).

Подамо на вхід лічильника ще один тактовий імпульс. Значення першого тригера знову зміниться на прямо протилежне. На цей раз на виході першого тригера, а значить і на вході другого тригера, сформується задній фронт. Це означає, що другий тригер теж змінить свій стан на протилежний. це чітко видно на часових діаграмах, наведених на рис. 4.2.

Продовжуючи аналізувати часову діаграму, можна визначити, що на виходах наведеної схеми лічильника послідовно з'являються цифри від 0 до 7. Ці цифри записані в двійковому вигляді. При надходженні на лічильний вхід лічильника чергового імпульсу, вміст його тригерів збільшується на 1. Тому такі лічильники отримали назву підсумовуючих лічильників. Якщо інформацію знімати з інверсних виходів тригерів, то вийде віднімаючий лічильник.

2.2. Лічильник з довільним модулем рахунку

Для побудови такого лічильника можна використовувати двійковий лічильник, у якого модуль рахунку М повинен бути більше модуля рахунку лічильника, що розробляється з довільним модулем рахунку.

Нехай потрібно зробити лічильник з М = 10. У чотирьохрозрядний лічильника модуль рахунку дорівнює 16 (більше 10). Схема лічильника являє собою 4 послідовно включених рахункових тригера, у яких є вхід скидання R (див. рис. 4.3). Число 10 в двійковій системі числення представляється 1010. Коли на виходах лічильника буде код 1010, на виході елемента «І» з'явиться логічна одиниця, яка запустить схему вимкнення. Тривалість імпульсу на виході схеми вимкнення повинна бути достатня для надійного скидання всіх тригерів лічильника в 0. Розряди числа 1010, що дорівнюють 1, подаються на схему «І» з прямих виходів тригерів, а рівні 0 – з інверсних.

Таким чином, як тільки лічильник дорахував до 10, відбудеться обнулення всіх тригерів, і рахунок продовжиться з коду 0000. Як схеми вимкнення може бути застосований RS-тригер. Сигнал на вході R лічильника буде діяти протягом одного періоду вхідних імпульсів.

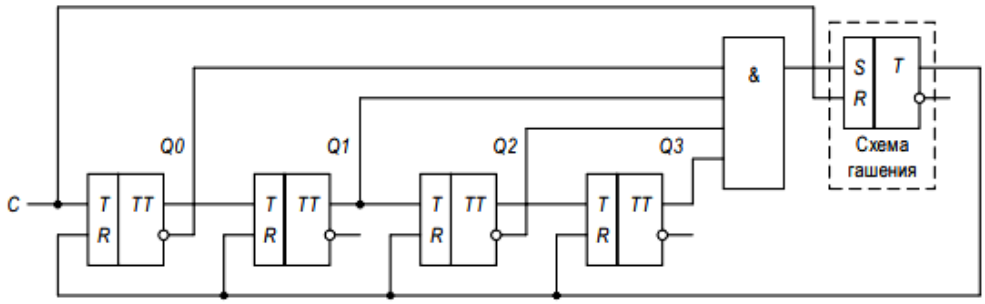


Рисунок 4.3 – Лічильник з модулем рахунку M = 10

Умовно-графічне позначення підсумовуючого довічного лічильника на принципових схемах наведено на рисунку 4.4. У лічильниках зазвичай передбачають вхід обнулення мікросхеми R, який дозволяє записати в усі тригери лічильника нульове значення. Цей стан іноді називають вихідним станом лічильника.

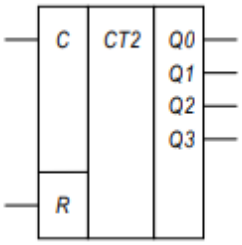


Рисунок 4.4 – Умовне графічне позначення чотирьохрозрядного двійкового лічильника

2.3. Кільцевій лічильник

У розглянутих схемах лічильників швидкодію всієї схеми визначається часом поширення сигналу від входу до виходу найстаршого розряду. При цьому виходить, що чим більше необхідний коефіцієнт ділення, тим більше двійкових розрядів лічильника потрібно для реалізації цього дільника. Тим більший час потрібно для поширення сигналу від входу синхронізації лічильника, до його виходу, і тим менше буде гранична частота, яку можна подавати на вхід цього дільника.

Щоб обійти цю неприємну особливість, потрібно, щоб лічильник готував свій новий стан в проміжках між тактовими імпульсами і тільки записував його по приходу нового імпульсу.

Проста схема, яка реалізує даний принцип, – це схема кільцевого лічильника. Такий лічильник можна побудувати на основі зсувного регістру. Схема кільцевого лічильника і часові діаграми його роботи наведені на рис. 4.5.

Як видно, при рахунку від першого розряду до останнього поширюється «хвиля» одиниць, а потім «хвиля» нулів. Для усунення цього ефекту вводиться додаткове комбінаційний пристрій (КУ) для формування правильних чисел

(Рис. 4.6).

Як перевага схеми кільцевого лічильника можна відзначити те, що її швидкодія залежить тільки від часу затримки одного тригера. Це означає, що на кільцевих лічильниках можна реалізовувати найшвидкодіючі подільники частоти.

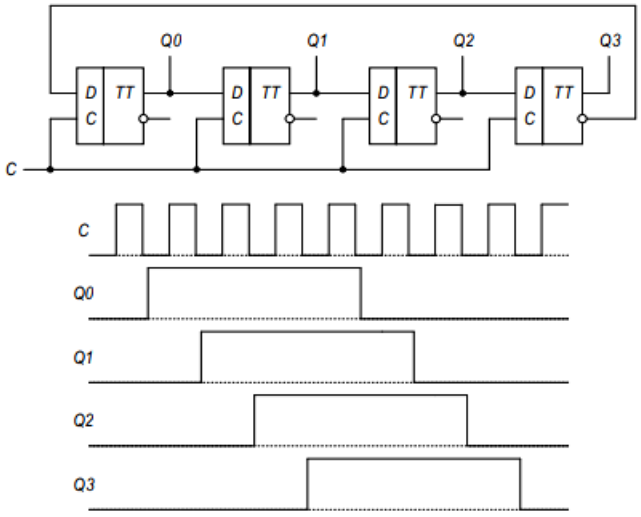


Рисунок 4.5 – Схема і часові діаграми кільцевого лічильника

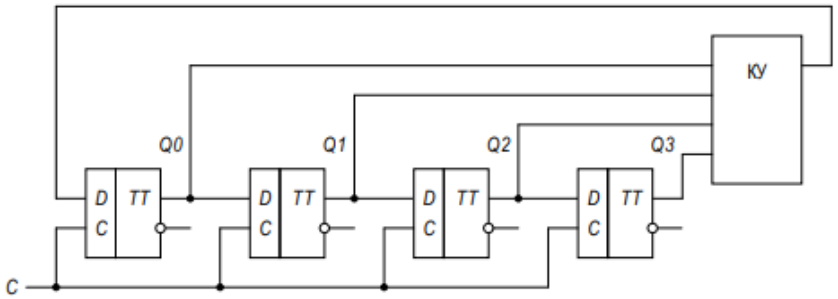


Рисунок 4.6 – Схема двійкового лічильника на основі кільцевого лічильника

3. Завдання до роботи

Запустити середовище Altera Quartus II і створити новий проект з параметрами:

– Ім'я проекту: LabWork_4_StudentName

– Сімейство ПЛІС: Cyclone IV E

– Тип ПЛІС: EP4CE115F29C7

Решту залишити за замовчуванням.

3.1. Дослідження роботи реверсивного лічильника в режимах підсумовування, віднімання і паралельної завантаження

1. Вивчити опис і режими роботи реверсивного лічильника SN74193 (вітчизняний аналог КР1533ІЕ7) в Додатку В.

2. Додати в проект новий файл схеми і накреслити схему лічильника на основі бібліотечного елемента «74193» (даний модуль є моделлю реальної

мікросхеми SN74193 / КР1533ІЕ7).

3. Виконати аналіз і синтез проекту і зробити призначення виводів відповідно до таблиці 4.1. Після цього виконати повну компіляцію проекту.

Таблиця 4.1

Призначення виводів лічильника

Сигнал	**G_50MHz**	**CU**	**CD**	**D3**	**D2**	**D1**	**D0**
Вивід ПЛІС	Y2	R24	N21	AD27	AC27	AC28	AB28
На стенді	50 MHz Oscillator	KEY3	KEY2	SW3	SW2	SW1	SW0

Сигнал	**LOAD**	**RESET**	**Q3**	**Q2**	**Q1**	**Q0**
Вивід ПЛІС	M21	M23	G21	G22	G20	H21
На стенді	KEY1	KEY0	LEDG7	LEDG6	LEDG5	LEDG4

4. Запрограмувати навчальний стенд і, спостерігаючи за станами світлодіодних індикаторів, проаналізувати роботу лічильника в трьох режимах – підсумовування, віднімання, паралельна завантаження. В останньому режимі на паралельний вхід лічильника необхідно завантажити число відповідно до свого варіантом (див. табл. 4.2).

3.2. Дослідження лічильника з довільним модулем рахунку

1. Змінити вихідну схему таким чином, щоб лічильник рахував до числа, визначеного варіантом (див. табл. 4.2).

2. Виконати повну компіляцію проекту.

3. Запрограмувати навчальний стенд і, спостерігаючи за станами світлодіодних індикаторів, переконатися в правильній роботі схеми.

4. Зміст звіту

1. Мета роботи.

2. Схема дослідження лічильника.

3. Часові діаграми вхідних і вихідних імпульсів в режимах підсумовування і віднімання.

4. Схема дослідження лічильника з довільним модулем рахунку.

5. Часові діаграми вхідних і вихідних імпульсів лічильника з довільним модулем рахунку.

6. Висновки по роботі.

5. Варіанти завдань

Таблиця 4.2

Варіанти завдань

Варіант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Число	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

6. Контрольні питання

1. Пояснити принцип роботи підсумовуючого лічильника.
2. Зобразити часові діаграми роботи підсумовуючого лічильника.
3. Пояснити принцип роботи віднімаючого лічильника.
4. Зобразити часові діаграми роботи віднімаючого лічильника.
5. Пояснити принцип роботи лічильника з довільним модулем рахунку.
6. У чому полягає перевага кільцевого лічильника перед асинхронними?
7. Назвіть основні параметри двійкових лічильників.
8. Де застосовуються лічильники?

Лабораторна робота №5
ВЕРИФІКАЦІЯ HDL-ПРОЕКТУ

1. Мета роботи

Метою роботи є вивчення можливостей верифікації HDL–проектів в середовищі ModelSim.

2. Короткі теоретичні відомості

Верифікація – в загальному випадку, це перевірка збігу результатів роботи розроблювального пристрою і вимог, які до нього пред'являються.

HDL-мови мають в своєму складі необхідний інструментарно для проведення верифікації та налагодження проектів ПЛІС. В якості середовища симуляції і налагодження пропонується використовувати программуModelSim, що входить до складу САПР AlteraQuartus II.

2.1. Паралельний оператор PROCESS в VHDL

Оператор *process* відноситься до класу операторів паралельної обробки, що дає можливість використовувати його для проектування цифрових пристроїв комбінаторного типу. У той же час він є батьківським оператором для всіх інших виконуваних операторів, які дозволено розташовувати в його тілі. Говорячи іншими словами, всі виконувані оператори, що розташовуються в тілі оператора *process*, виконуються по послідовному принципу. Оператор *process* може використовуватися як в явному вигляді, так і в неявному.

Явно заданий оператор *process* – це основна конструкція для поведінкової форми опису проектів. Розрізняють два підкласу явно заданого оператора ***process*** : зі списком чутливості і без списку чутливості.

Список чутливості (*sensitivitylist*) – це набір сигналів, зміна значення яких викликає негайний запуск на виконання оператора *process*. Якщо список чутливості не визначений, всередині такого процесу повинен міститися оператор *wait*, щоб надати розробнику можливість управляти запуском і зупинкою відповідного оператора *process* . Оператора без списку чутливості завжди запускається на виконання автоматично в момент початку процесу моделювання роботи проекту.

Синтаксис явно заданого оператора *process* має наступний вид (в квадратних дужках вказані необов'язкові конструктивні елементи):

[мітка_процесу:] **process** [(список_чутливості)] [**is**]
[оператори_об'явлення_процесу] *– розділ оголошень*
Begin*– розділ виконуваних операторів*
– Тут розміщуються оператори наступних типів:
– установки значень сигналів;
– присвоєння значень змінним;
– виклики процедур;

– оператори ***case, exit, if, loop, next, null*** *або* ***wait*** *;*

endprocess [мітка_процесу];

Необхідно відзначити, що присвоєння значення змінної в тілі оператора *process* позначається символами «: =», а встановлення значення сигналу символами «<=».

2.2. Послідовні оператори VHDL

Оператор *if* відноситься до сімейства послідовних операторів, що розташовуються в розділі виконуваних операторів явно заданого процесу. Оператор *if* відповідальний за виконання того чи іншого блоку послідовних операторів в залежності від істинності одного або декількох умов (condition). Синтаксис оператора *if* має такий вигляд:

```
if умова_1 then
        блок_операторів_1;
[ elsif умова_2 then
        блок_операторів_2;]
...
[ else
        блок_операторів_3;]
endif;
```

Оператор *wait* призупиняє виконання тіла оператора process до тих пір, поки не відбудеться ще одна обставина.

Мова VHDL підтримує кілька форм оператора *wait* , а саме:

waituntil умова; - *чекати виконання умови*

waitfor час **;** - *чекати заданий час, наприклад,* ***25ns***

waiton сигнал; - *чекати змінення значення сигналу*

wait; - *нескінченне очікування*

2.3. Testbench

Testbench – це віртуальний «стенд тестування» цифрової схеми. Створюється середовище, всередині якого поміщається тестованасхема як компонент. Тобто файл Testbench – це такий же HDL-файл, компонентом якого є тестована схема, описана на HDL-мові.

Усередині цього середовища описується генерація вхідних сигналів для схеми, які потім подаються на вхід для подальшого аналізу виходів тестованої схеми. При цьому сам Testbench не має зовнішнього інтерфейсу.

Потім цей файл симулюється в програмі симуляторі, яка буде виконувати

дії, описані в цьому файлі і представляти результат в графічному вигляді у вигляді діаграм зміни сигналів.

Змінним параметром буде час – симулятор послідовно буде збільшувати час з необхідним дозволом і послідовно обчислювати значення сигналів.

Приклад Testbench-файлу, описаного на VHDL:

```
library ieee;
use ieee.std_logic_1164.all;

entity temp_vhd_tst is
end temp_vhd_tst;

architecture temp_arch of temp_vhd_tst is
        signal x1: std_logic;
        signal x2: std_logic;
        signal x3: std_logic;
        signal y: std_logic;
- оголошення, що тестується схема
component temp
        port (
                x1: in std_logic;
                x2: in std_logic;
                x3: in std_logic;
                y: out std_logic
        );
end component;
begin
        - вставка тестованої схеми
        i1: temp
        - зв'язок між портами і сигналами
        port map (
                x1 => x1,
                x2 => x2,
                x3 => x3,
                y => y
        );

        init: process
        begin
                - одноразово виконується код
                wait ;
        end process init;

        always: process
        - список чутливості
        begin
        - код, що виконується після зміни будь-якого з сигналів
        зі списку чутливості
```

```
        end process always;
end temp_arch;
```

2.4 Оператори перевірки умов і виведення повідомленьв консоль

Мова VHDL допускає вбудовування в файл Testbench перевіркувиконання певних умов. Це робиться за допомогою***оператора assert*** . Приклад його використання показаний нижче:

```
assert SUM_EXPECTED = SUM_OUT - умова для перевірки
report "ERROR: output SUM isincorrect" severity warning;
```

Після ключового слова *assert* вказується умова, якапередбачається істинним (при нормальній роботі модуля воно повинна виконуватися). Якщо умова не виконується, в консоль виводиться повідомлення, вказане після ключового слова ***report***. Після *severity*вказується рівень важливості перевіряють умови.

Рівень важливості може бути наступним:

• *note* - «примітка», це повідомлення не є помилкою і носить інформаційний характер;

• *warning* - «попередження», такі повідомлення вираховуються, але їх наявність не є підставою для визнання результатів некоректними;

• *error* - «помилка», результат моделювання визнається некоректним, моделювання триває;

• *failure* - «збій», аналогічно до помилки, але виконання негайно переривається.

За замовчуванням повідомлення мають рівень *error* .

Призначення рівня важливості проводиться самим розробником тесту. Передбачається, що повідомлення рівня «збій» будуть привласнювати помилки, які не можуть бути досить легко виправлені, або відносяться до «ситуацій, які ніколи не можуть виникнути ». Виникнення повідомлень такого рівня має означати, що в процесі моделювання виникли істотні проблеми, і виробляти подальший аналіз не має сенсу.

3. Завдання до роботи

3.1. Симуляція проекту вручну

1. Запустити середовище AlteraQuartusII і створити проект схеми у відповідності зі своїм варіантом (див. табл. 5.1). Слід мати на увазі, що ModelSim не підтримує симуляцію файлів схем, тому краще описати схему на одному з HDL-мов.

Примітка. Для симуляції схеми потрібно перетворити її в HDL-опис (File

→ Create / Update → Create HDL FilefromCurrentFile). У вікні, вибрати будь-яку зручну мову. Відкрити отриманий файл і додати його в проект (Project → AddCurrentFileto Project). Призначити його файлом верхнього рівня (Project → SetasTop-LevelEntity), а файл зі схемою (розширення * .bdf) видалити з проекту (Project → Add / RemoveFilesin Project).

2. Виконати аналіз і синтез проекту, призначити висновки і повну компіляцію проекту. У разі успішної компіляції, запустити симуляцію проекту (Tools → RunSimulationTool → RTL Simula-tion). При цьому слід вибрати мову, що відповідає файлу верхнього рівня проекту. AlteraQuartus II запустить програму ModelSim.

3. У вікні ModelSim запустити симуляцію свого проекту (Simulate → StartSimulation) і вибрати файл верхнього рівня проекту в каталозі «work». На панелі Objects з'являться порти і сигнали, наявні в проекті. Виділити вхідні та вихідні порти і перетягнути їх, захопивши мишею, в панель Wave.

4. Поставити сигнали на вхідні порти таким чином, щоб вони утворювали інкрементування у двійковий код (Wave → Clock ...), параметр FirstEdge задати «Falling». Встановити загальний час симуляції, достатню для симуляції всіх можливих станів входів, і запустити симуляцію (Simulate → Run → Run 100).

5. Програма проведе симуляцію заданого часу роботи схеми і відобразить результат у вигляді часової діаграми на панелі Wave. Перевірити відповідність діаграми таблиці істинності заданої функції.

6. Завершити роботу програми ModelSim.

3.2. Testbench

1. У AlteraQuartus II виконати генерацію шаблону файлу Testbench (Processing → Start → StartTestBenchTemplateWriter). На панелі Messages знайти рядок з інформацією про розташування згенерованого файлу (див. рис. 5.1).

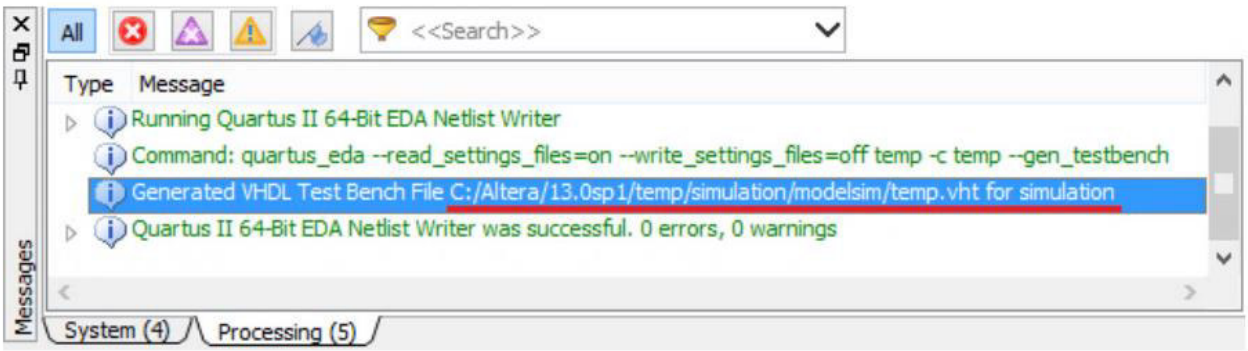


Рис. 5.1 - Приклад повідомлення Quartus II про розташування файлу Testbench

2. Виконати прив'язку файлу Testbench до проекту. Для цього відкрити налаштування проекту (Assignments → Settings), розділ Simula- tion і додати

новий Testbench, виконуючи вказівки малюнка 5.2.

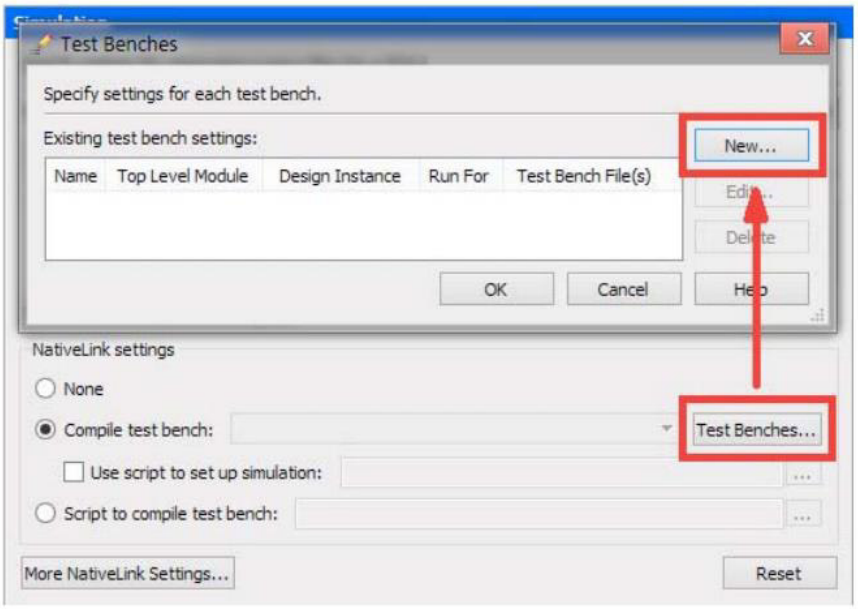


Рис. 5.2 - Налаштування симуляції, додавання Testbench

У вікні налаштувань (рис. 5.3) ввести:

• Testbenchname: довільне ім'я;

• Toplevelmoduleintestbench: ім'я файлу Testbench верхнього рівня (вказано в початковому тексті файлу Testbench як ім'я об'єкта *entity*);

• Simulationperiod: період симуляції відповідно до п. 3.1;

• Testbenchandsimulationfiles: вибрати Testbench-файл і нажати кнопку Add.

3. Відкрити Testbench-файл в середовищі AlteraQuartus II (File →Open) і ввести код, що описує оточення для тестування проекту: генерація вхідних сигналів і висновок консольних повідомлень про початок і завершення тесту.

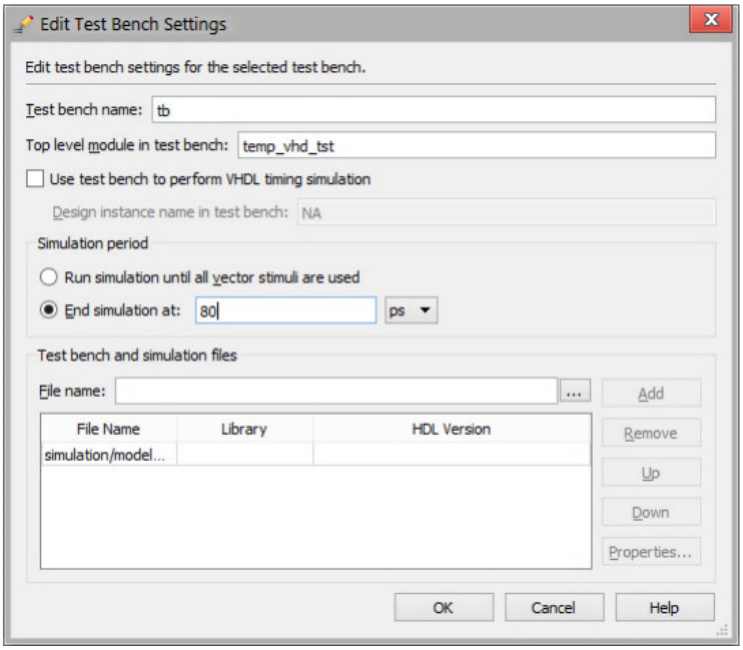


Рис. 5.3 - Налаштування Testbench

4. Виконати компіляцію проекту. У разі успішної компіляції запустити симуляцію проекту (Tools → RunSimulationTool → RTL Simulation).

5. Результат симуляції відобразиться у вигляді діаграми на панеліWave. Текстові повідомлення виводяться в панелі Transcript. Впевнитись, що виводиться повідомлення про закінчення тестування.

6. Закінчити роботу програми ModelSim.

4. Зміст звіту

1. Мета роботи.
2. HDL-опис схеми і її таблиця істинності за спрощеним варіантом.
3. Часова діаграма симуляції.
4. Вихідний код файлу Testbench.
5. Висновки по роботі.

5. Варіанти завдань

Таблиця 5.1

Варіанти завдань

Варіант	**1**	**2**	**3**	**4**	**5**
Функція	$A(B+C)$	$A+BC$	$\overline{ABC}$	$\overline{A}+\overline{BC}$	$A\oplus B\oplus C$
Варіант	**6**	**7**	**8**	**9**	**10**
Функція	$\overline{A\oplus B\oplus C}$	ABC	$A+B+C$	$\overline{AB+C}$	$\overline{A+B+C}$
Варіант	**11**	**12**	**13**	**14**	**15**
Функція	$A\oplus BC$	$\overline{A+B\oplus C}$	$\overline{A}B+C$	$\overline{A+B+C}$	$\overline{AB\otimes C}$

6. Контрольні питання

1. Що таке симуляція проекту? Для чого вона проводиться?

2. Поясніть принцип тестування проектів для ПЛІС із використанням файлів Testbench.

3. Опишіть структуру файлу Testbench. Чим він відрізняється від VHDL-файлу?

4. Що таке компонент в VHDL? Як він реалізується?

5. Поясніть роботу оператора *process*. Яке його призначення в мові VHDL?

6. Як працює оператор умови в мові VHDL? Наведіть приклад використання оператора *if*.

7. Наведіть приклад опису періодичного сигналу.

8. Як вивести повідомлення при тестуванні проекту?

Лабораторна робота №6

Вивчення архітектури та принципа роботи RISC-процесора Nios II

1. Мета роботи

Метою роботи є вивчення архітектури і системи команд процесора Nios II, а також придбання навичок його програмування на мові асемблера.

2. Короткі теоретичні відомості

Процесор Nios II - це 32-розрядна процесорна RISC архітектура для застосування у вбудованих системах (soft processor або програмний процесор), розроблена спеціально для ПЛІС фірми Altera. Nios II є розвитком архітектури Nios і знаходить застосування в різних вбудованих додатках - від систем цифрової обробки сигналів до різноманітних пристроїв управління.

Існують 3 різні версії конфігурації процесора.

• *Nios II / f (fast)* - версія, призначена для досягнення максимальної продуктивності. Конфігурація має широкий набір опцій для оптимізації процесора по продуктивносі.

• *Nios II / s (standart)* - стандартна версія. Дана конфігурація вимагає менше ресурсів для реалізації і характеризується меншою продуктивністю.

• *Nios II / e (economy)* - економічна версія. Дана конфігурація вимагає найменшу кількість ресурсів для реалізації, але володіє обмеженим набором можливостей.

2.1. Архітектура процесора Nios II

Процесор Nios II має RISC архітектуру, в якій всі арифметичні і логічні операції виконуються над операндами, що зберігаються в регістрах загального призначення. Обмін інформацієюміж регістрами і пам'яттю здійснюється шляхом виконання спеціальних команд «Load» і «Store».

Довжина машинного слова процесора Nios II становить 32 біта. Таку ж розрядність мають і його регістри. процесор використовує роздільні шини для команд і даних, тобто побудований за гарвардської архітектури. Структурна схема процесора Nios II представлена на рис. 6.1.

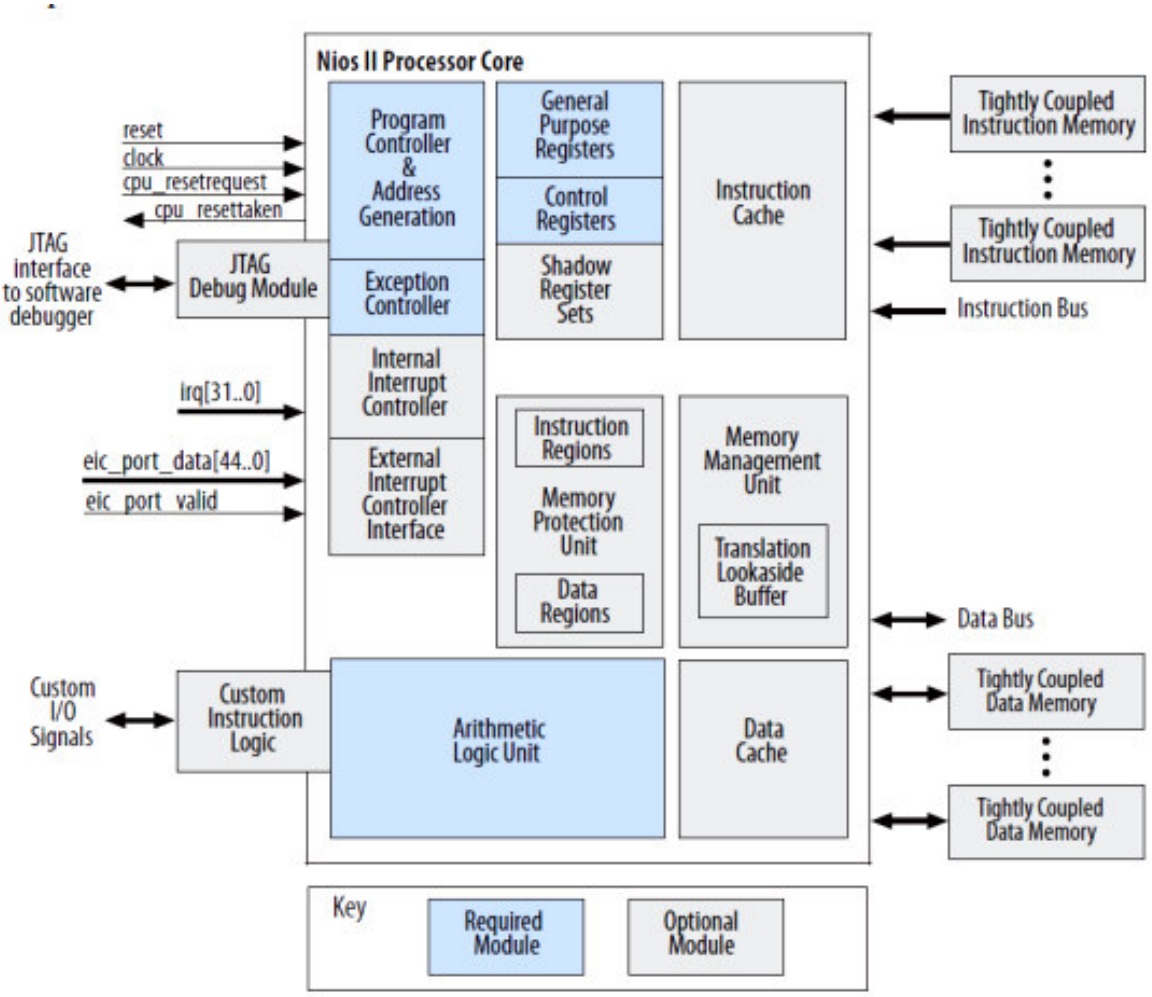


Рис. 6.1 - Структурна схема процесора Nios II

Процесор може функціонувати в двох режимах - в режимі супервізора (процесор перемикається в даний режим після надання сигналу скидання), де йому дозволяється виконувати всі інструкції і здійснювати будь-які функції, і в режимі користувача, в якому обмежена можливість виконання певних інструкцій системного призначення (доступний тільки при наявності модуля управління пам'яттю MMU або модуля захисту пам'яті MPU).

Архітектурою процесора Nios II підтримується однорідний регістровий файл, що складається з тридцяти двох регістрів загального назначення (РОН) і до тридцяти двох керуючих регістрів. Всі регістри мають імена, які розпізнаються асемблером. Регістри загального призначення і керуючі регістри, представлені в таблицях 6.1 і 6.2, відповідно. До складу процесора може входити до 32 управляючих регістрів. Їх загальна кількість залежить від наявності модулів захисту пам'яті або управління пам'яттю.

Таблиця 6.1

Регістри загального призначення процесора Nios II

Регістр	**Ім'я**	**Призначення**
r0	zero	Регістр нуля, завжди містить значення 0x00000000
r1	at	Часовий регістр, використовується асемблером
r2, r3	v0, v1	Повертання значення
r4..r7	a0, a3	Регістр аргумента
r8..r15	t0..t7	Часові регістри
r16..r23	s0..s7	Збережні регістри
r24	et	Часовий регістр для обробки винятків
r25	bt	Часовий регістр, використовуваний при налагодженні
r26	gp	Глобальний покажчик
r27	sp	Покажчик стека
r28	fp	Покажчик кадра
r29	ea	Адреса повернення з винятків (недоступний в режимі користувача)
r30	ba	Повернення з контрольної точки (використовується тільки модулем налагодження JTAG)
r31	ra	Адреса повернення при виклику підпрограм

Опціонально, для прискорення контекстних перемикань, процесор може мати до 63 наборів тіньових (shadows) регістрів. Для роботи з ними процесор має дві спеціальні інструкції, що дозволяють переміщати дані між наборами регістрів. Поточний використовуваний набір тіньових регістрів відображає поле CRS статусного регістра.

Для адресації процесор Nios II використовує 32-бітову адресу з побайтовою адресацією (порядок Little-endian). За допомогою відповідних команд можна записувати або зчитувати слова (32 біта), півслова (16 біта) і байти даних (8 біт).

У процесорі Nios II визначені кілька способів адресації. *Безпосередня адресація* - в команді присутня 16-бітний операнд (він може доповнюватися до 32 розрядів при виконанні арифметичних операцій); *реєстрова адресація* - операнди знаходяться в регістрах процесора; *відносна реєстрова адресація*- ефективний адреса операнда обчислюється підсумовуванням вмісту регістра і знакового 16-розрядного

числа, що знаходиться в самій команді, яка визначає зміщення; *непряма регістроваадресація* - вміст регістра є ефективним адресом операнда (цей спосіб еквівалентний попередньому при нульовому зміщенні), *абсолютна адресація* - 16-бітний абсолютний адреса операнда визначено зміщенням щодо регістра нуля.

Таблиця 6.2

Керуючі регістри процесора Nios II

Регістр	**Ім'я**	**Призначення**
ctl0	status	Регістр стану: біт 0 PIE - дозвіл зовнішніх переривань, біт 1 U - режим роботи процесора, біти 2..32 - зарезервовані
ctl1	estatus	Копія регістра стану під час переривань: біт 0 EPIE - збережений біт PIE, біт 1 EU - збережений біт U, біти 2..32 - зарезервовані
ctl2	bstatus	Копія регістра стану під час точок зупинки: біт 0 PIE - збережений біт PIE, біт 1 U - збережений біт U, біти 2..32 - зарезервовані
ctl2	bstatus	Регістр дозволу переривань
ctl3	ienable	Регістр активних переривань
ctl5	cpuid	Унікальний ідентифікатор процесора для багатопроцесорних систем

Процесор містить апаратну обробку винятків, включно апаратні переривання. Також він може мати додатковий інтерфейс із зовнішнім контролером переривань (EIC), щоб прискорити обробку переривань в комплексній системі.

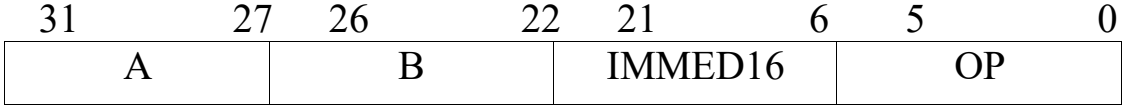
До складу процесорного ядра також можуть входити і спеціалізовані модулі, наприклад, JTAG UART, який служить для обмін інформацією з комп'ютером, і JTAG Debug, необхідний для виконання налагодження програмного забезпечення за допомогою комп'ютера.

2.2. Система команд процесора Nios II

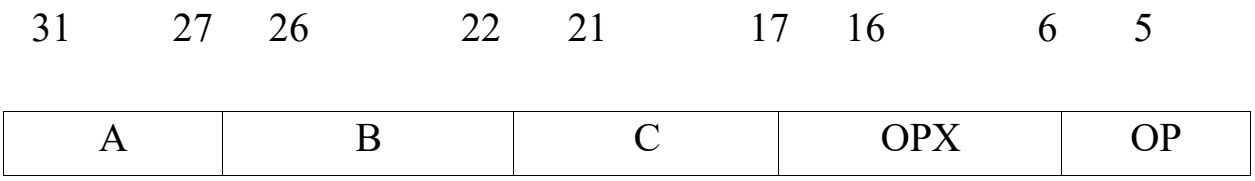
Виконані команди процесора Nios II кодуються 32-розрядними

словами. Використовуються 3 різні формату команд:

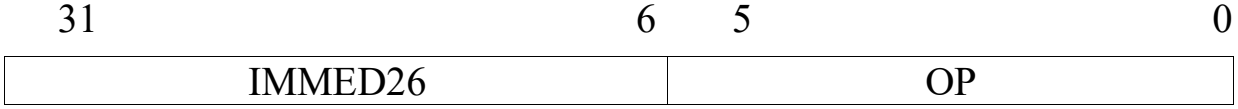
• **I-тип** . Команда містить поля A і B шириною по 5 біт, використовуючи для визначення регістрів, поле IMMED16, що використовується для вказівки безпосередніх операндів (при необхідності може бути розширено до 32 біт), і поле коду операції (КОП, OP) шириною 6 біт;



• **R-тип** . Команда містить поля A, B і C шириною по 5 біт,використовуються для визначення регістрів, поле OPX длярозширення коду операцій, і поле коду операції (КОП, OP)шириною 6 біт;



• **J-тип** . Команда містить поле IMMED26, що використовується длявказівки безпосередніх операндів, і поле коду операції(КОП, OP) шириною 6 біт.



2.3. Список команд процесора Nios II

Нижче наведено неповний список команд асемблера процесора Nios II.

Команди load, store

Для обміну інформацією між регістрами загального призначення процесора і оперативною пам'яттю визначено кілька команд.

ldw *rB, byte_offset (rA)* - завантаження слова з оперативної пам'яті в регістр rB, адреса операнда в оперативній пам'яті визначається складеним вмісту регістра rA і зміщення byte_offset;

stw *rB, byte_offset (rA)* - збереження слова з регістра rB в оперативну пам'ять за адресоюrA + зміщення byteoffset.

Арифметичні команди

***add** rC, rA, rB* і ***addi** rB, rA, IMMED* - складена; у першому випадку

обидва операнда (rA, rB) розташовані в регістрах загального призначення, у другому випадку один з операндів (IMMED) представлений числом, результат обчислення в обох випадках поміщається в РОН (rC);

***sub** rC, rA, rB* - віднімання; команда віднімання константи замінюється операцією додавання з негативною константою (-IMMED);

***mul** rC, rA, rB* - множення;

***mulxss** rC, rA, rB* - множення двох знакових чисел;

***mulxsu** rC, rA, rB* - множення знакового і беззнакового чисел;

***mulxuu** rC, rA, rB* - множення двох беззнакових чисел;

Логічні команди

Наступні пари команд призначені для виконання логічних операцій.

***and** rC, rA, rB* і ***andi** rD, rA, IMMED16* - побітове логічне І;

***or** rC, rA, rB* і ***ori** rB, rA, IMMED16* - побітове логічне АБО;

***xor** rC, rA, rB* і ***xori** rB, rA, IMMED16* - побітове додавання по модулю 2 (виключає АБО);

***nor** rC, rA, rB* - побітове логічне І з запереченням.

Додаткові три команди ***andhi*** , ***orhi*** , ***xorhi*** виконуютьоперації зі старшою половиною слова в регістрі. Молодша частина безпосереднього операнда при цьому доповнюється нулями до повної ширини слова.

Команди зсуву

Команди логічного зсуву виконують зсув вмісту регістра rA вліво або вправо на кількість розрядів, заданих або п'ятьма молодшими розрядами регістра rB, або безпосереднім операндом IMMED5. Вивільнені розряди заповнюються нулями. Результат заноситься в регістр rC.

***sll** rC, rA, rB* і ***slli** rC, rA, IMMED5* - логічний зсув вліво;

***srl** rC, rA, rB* і ***srli** rC, rA, IMMED5* - логічний зсув вправо;

Команди арифметичного зсуву виконують здвигвмісту регістра rA вправо на кількість розрядів, заданих або п'ятьма молодшими розрядами регістра rB, або безпосереднім операндом IMMED5. Вивільнені розряди заповнюються знаковимбітом. Результат заноситься в регістр rC.

***sra** rC, rA, rB* і ***srai** rC, rA, IMMED5* - арифметичний зрушеннявправо.

Команди циклічного зсуву виконують зсув вмістурегістра rA вліво або вправо на кількість розрядів, заданих абоп'ятьма молодшими розрядами регістра rB, або безпосереднімоперандом IMMED5. Вивільнені розряди заповнюються витісненими битами. Результат заноситься в регістр rC.

ror *rC, rA, rB* - циклічний зсув вправо;

rol *rC, rA, rB* і ***roli*** *rC, rA, IMMED5* - циклічний зсув вліво;

Команди пересилання

Для копіювання даних з одного регістра в інший або записи константи в РОН можна застосувати команди складання add, вказавши в якості другого операнда регістр нуля, або addi, вказавши нулеву константу.

Команди порівняння

Група команд порівняння призначена для порівняння вмісту регістрів.

cmplt *rC, rA, rB* - запис в регістр rC одиниці, якщо rA<rB (з урахуванням знака), або нуля в іншому випадку;

cmpltu *rC, rA, rB* - запис в регістр rC одиниці, якщо rA<rB(без урахування знака), або нуля в іншому випадку;

cmpeq *rC, rA, rB* - запис в регістр rC одиниці, якщо rA = rB,або нуля в іншому випадку;

cmpne *rC, rA, rB* - запис в регістр rC одиниці, якщо rA ≠ rB,або нуля в іншому випадку;

cmpge *rC, rA, rB* - запис в регістр rC одиниці, якщо rA ≥ rB(з урахуванням знака), або нуля в іншому випадку;

cmpgeu *rC, rA, rB* - запис в регістр rC одиниці, якщо rA ≥ rB(без урахування знака), або нуля в іншому випадку.

Команди порівняння з безпосереднім операндом

Наступна група команд призначена для порівняння вмісту регістра з безпосереднім операндом.

cmplti *rB, rA, IMMED16* - запис в регістр rC одиниці, якщо rA<IMMED16 (з урахуванням знака), або нуля в іншому випадку;

cmpltui *rB, rA, IMMED16* - запис в регістр rC одиниці, якщо

rA<IMMED16 (без урахування знака), або нуля в іншому випадку;

cmpeqi *rB, rA, IMMED16* - запис в регістр rC одиниці, якщо rA = IMMED16, або нуля в іншому випадку;

cmpnei *rB, rA, IMMED16* - запис в регістр rC одиниці, якщо rA ≠ IMMED16, або нуля в іншому випадку;

cmpgei *rB, rA, IMMED16* - запис в регістр rC одиниці, якщо rA ≥ IMMED16 (з урахуванням знака), або нуля в іншому випадку;

cmpgeui *rB, rA, IMMED16* - запис в регістр rC одиниці, якщо rA ≥ IMMED16 (без урахування знака), або нуля в іншому випадку.

Команди переходів

Команди переходів застосовуються для розгалуження програми.

jmp *rA* - безумовний перехід за адресою в rA;

jmpi *Label* - безумовний перехід за адресоюLabel, адреса переходу обчислюється як PC 31..28 : IMMED26 x 4;

br *Label* - безумовний перехід заадресоюLabel, адреса переходу обчислюється як PC + 4 + σ (IMM16);

blt *rA, rB, Label* і ***bltu*** *rA, rB, Label* - команди переходу по адресу Label, якщо rA<rB, з урахуванням і без урахування знака відповідно;

bge *rA, rB, Label* і ***bgeu*** *rA, rB, Label* - команди переходу по адресу Label, якщо rA ≥ rB, з урахуванням і без урахування знака відповідно;

beq *rA, rB, Label* - перехід за адресоюLabel, якщо rA = rB.

bne *rA, rB, Label* - перехід за адресоюLabel, якщо rA ≠ rB.

Команди виклику підпрограми і повернення з неї

call *Label* - адреса наступної команди зберігається в регістрі rai виконується перехід, адреса переходу обчислюється як $PC_{31..28}$: IMMED26 x 4.

callr *rA* - адреса наступної команди зберігається в регістрі ra і у разі переходу за адресою, що зберігається в регістрі rA.

ret -повернення з підпрограми, у разі переходу за адресою, що зберігається в регістрі rA.

Команди управління

Запис в регістри управління процесора Nios II або читання з них виконуються за допомогою наступних команд:

rdctl *rC, ctlN* - копіювання вмісту регістра ctlN в rC;

wrctl *ctlN, rA* - копіювання вмісту регістра rA в ctlN.

3. Завдання до роботи

1. Скласти блок-схему алгоритму програми відповідно до варіанту завдання.

2. Запустити програму Nios2Sim і ввести код програми на мові асемблера. Вихідний масив оголосити в пам'яті програми. Результат виводити в оперативну пам'ять.

3. Запустити код на виконання (меню Nios II → StartSimulation) і виконати покрокове виконання програми (менюNios II → Execute a Step), спостерігаючи за станом пам'яті або регістрів.

4. Зміст звіту

1. Мета роботи.

2. Завдання до роботи схеми.

3. Блок-схема алгоритму програми.

4. Вихідний код програми.

5. Висновки по роботі.

5. Варіанти завдань

1. У масиві всі парні елементи обнулити.

2. У масиві всі непарні елементи замінити на цифру 1.

3. У масиві всі елементи, які стоять після непарних, обнулити.

4. У масиві всі елементи, які стоять перед парними, замінити на цифру 9.

5. У масиві всі елементи, які стоять між перед мінімальним,замінити на 0.

6. У масиві всі елементи, які стоять після мінімального, замінити на цифру 7.

7. У масиві всі елементи, які стоять перед максимальним, замінити на 0.

8. У масиві всі елементи, які стоять після максимального, замінити на 0.

9. У масиві всі непарні елементи, які стоять після максимального,

замінити на 0.

10. У масиві всі парні елементи, які стоять перед мінімальним, замінити на 0.

11. У масиві всі непарні елементи по модулю більші 5, і стоять після максимального, замінити на 0.

12. У масиві всі парні елементи по модулю більші 4, і стоять перед мінімальним, замінити на 0.

13. У масиві все елементи, кратні 2 і 3, обнулити.

14. Відсортувати масив по спадаючій.

15. Відсортувати масив по зростанню.

Масив повинен мати розмірність більше ніж 1 × 20 і бути заповнений довільними числами.

6. Контрольні питання

1. Що таке Nios II? Що таке система з процесором Nios II?

2. Які існують конфігурації процесора NiosII? У чому їх відмінність?

3. Що таке регістровий файл? Поясніть його структуру і призначення.

4. Назвіть види адресації процесора Nios II.

5. Назвіть формати команд процесора Nios II з прикладами.

6. Якими командами мови асемблера здійснюється обмін інформацією між регістрами і пам'яттю?

7. Як на мові асемблера організовуються цикли?

8. Як на мові асемблера організовуються умовні конструкції?

СПИСОК ЛІТЕРАТУРИ

1. Цифрова схемотехніка. Навчальний посібник./ М.Г. Лорія, П.Й. Єлісєєв, О.Б. Целіщев. – Сєвєродонецьк: Вид-во Східноукр. нац. ун-ту імені Володимира Даля, 2016. – 280 с., 112 іл., 9 табл., 30 бібліогр. назв.
2. Мірошник М. А., Клименко Л. А., Корольова Я. Ю. Технології та автоматизація проєктування цифрових пристроїв складних комп'ютерних систем на ПЛІС: Навч. посібник. – Харків: УкрДУЗТ, 2021. – 220 с.
3. Основи та методи цифрової обробки сигналів: від теорії до практики: навч. посібник / Ушенко Ю.О., М.С. Гавриляк, М.В. Талах, В.В. Дворжак. – Чернівці : Чернівецький нац. ун-т ім. Ю. Федьковича. – 2021. – 308 с.
4. Лахно В.А., Гусєв Б.С., Смолій В.В., Місюра М.Д., Касаткін Д.Ю. Технології проектування комп'ютерних систем (частина 1) - К.: НУБіП України, 2019. – 205 с.
5. Alessio S.M. Digital Signal Processing and Spectral Analysis for Scientists, 1-st Edition - Switzerland: Springer Cham, 2016. – 924p.
6. Ефремов, Н.В. Введення в систему автоматизованого проектування Quartus II: навч. посібник / Н.В. Єфремов. – К.: Вища школа, 2011. – 147с.
7. Соловйов, В.В. Основи мови проектування цифрової апаратури Verilog / В.В. Соловйов. – К.: Вища школа, 2014. – 206с.

ДОДАТОК А

ІДЕНТИФІКАТОР МОВ VHDL І VERILOG

Ідентифікатори – визначені користувачем слова, використовуючи для позначення програмних об'єктів (проектів, інтерфейс, архітектур, пакетів, змінних, компонентів, сигналів, констант і ін.).

У мові VHDL при виборі ідентифікаторів слід дотримуватися такі правила:

• Ідентифікатори не повинні збігатися з ключовими словами;

• Ідентифікатор може містити прописні і (або) рядкові символи *латинського* алфавіту і цифри, розділені *одним* символом підкреслення;

• Першим символом в ідентифікатор повинен бути символ латинського алфавіту. Символ підкреслення не може бути першим або останнім символом ідентифікатора;

• Використовувати поспіль два символу підкреслення в ідентифікатор забороняється;

• Малі та великі символи латинського алфавіту в ідентифікаторі вважаються однаковими, т. е. ідентифікатори «And2», «AND2» або «and2» рівнозначні;

• Довжина ідентифікатора за стандартом не обмежена, але може обмежуватися в конкретному компіляторі.

Для мови Verilog існує ще кілька обмежень:

• Системні імена починаються з символу ($);

• Директиви компілятора починаються з символу '(апостроф);

• Компілятор розрізняє малі та великі літери.

Приклади коректних ідентифікаторів: X10, x_10, My_gate1.

Приклади некоректних індикаторів: _X10, gate @ inp, 2gate-in.

ДОДАТОК Б
ПРИДУШЕННЯ БРЯЗКІТУ КОНТАКТІВ

У навчальному стенді Altera DE2–115 для введення даних використовуються кнопки і перемикачі. Якщо перемикачі, як правило, використовуються для завдання статичного рівня, то кнопка використовується в якості генератора коротких імпульсів. У цьому випадку схема повинна реагувати не на статичний рівень, а на факт натискання, тобто на перехід сигналу з одного логічного стану на виведення ПЛІС в інше. При використанні механічних комутаторів (кнопки, тумблери і інші ключі) завжди виникає таке явище,як брязкіт контактів.

Брязкіт контактів – явище, яке виникає в електричних і електронних перемикачах, при якому вони замість деякого стабільного перемикання виробляють випадкові багаторазові неконтрольовані замикання і розмикання контактів (відбувається в момент перемикання) (див. рис. В.1). Іншими словами – це явище,викликане неминучим недосконалістю технології виготовлення перемикачів.

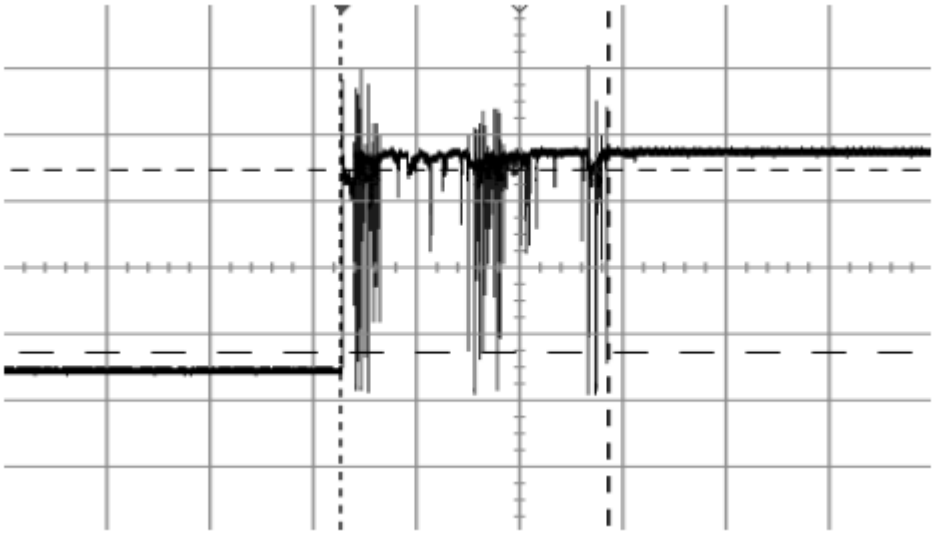


Рисунок В.1 – Брязкіт контактів на екрані осцилографа

Нижче наведено лістинг коду універсального модуля обробки сигналу з механічним перемикачем для придушення ефекту брязкоту контактів. В іноземній літературі аналогічний прилад або блок називається «debouncer». Модуль написаний на мові Verilog.

```
module button_debouncer
# (
        parameter CNT_WIDTH = 16
```

```
)
(
        input  clk_i,
        input  rst_i,
        input  sw_i,
        outputreg  sw_state_o,
        outputreg  sw_down_o,
        outputreg  sw_up_o
);

reg  [1: 0] sw_r;
always  @ ( posedge  rst_i  or  posedgeclk_i)
        if  (rst_i)
                sw_r<= 2'b00;
        else
                sw_r<= {sw_r [0], ~ sw_i};

reg  [CNT_WIDTH-1: 0] sw_count;

wire  sw_change_f = (sw_state_o! = sw_r [1]);
wire  sw_cnt_max = &sw_count;

always  @ ( posedge  rst_i  orposedge  clk_i)
        if  (rst_i)
        begin
                sw_count<= 0;
                sw_state_o<= 0;
        end
        else  if  (sw_change_f)
        begin
                sw_count<= sw_count + 'd1;
                if (sw_cnt_max) sw_state_o<= ~ sw_state_o;
        end
        else  sw_count<= 0;
```

```
always @( posedge clk_i)
begin
        sw_down_o<= sw_change_f&sw_cnt_max& ~ sw_state_o;
        sw_up_o<= sw_change_f&sw_cnt_max&sw_state_o;
end

endmodule
```

Для введення сигналу безпосередньо з кнопки є вхіднийпорт *sw_i* . Вхід для системних тактових імпульсів, з якими синхронно працює вся інша схема - *clk_i* . Вхід для асинхронного скидання всієї схеми - *rst_i* . З метою універсальності передбачено два роздільних виходу для події «кнопка натиснута» і події «кнопка відпущена» - *sw_down_o* і *sw_up_o* . Вихід *sw_state_o* служитьдля відображення стану кнопки (тривалості натискання), природньо, після усунення ефекту брязкоту.

Приклад підключення схеми придушення брязкоту контактів кнопки показаний на рис. В 2. Сигнал з кнопки повинен бути пропущенийчерез модуль (подається на вхід *BUTTON*), замість того, щоб йти безпосередньо до досліджуваної схеми. Крім цього, на модуль повинен подаватися тактовий сигнал з генератора (вхід *G_50MHz*).

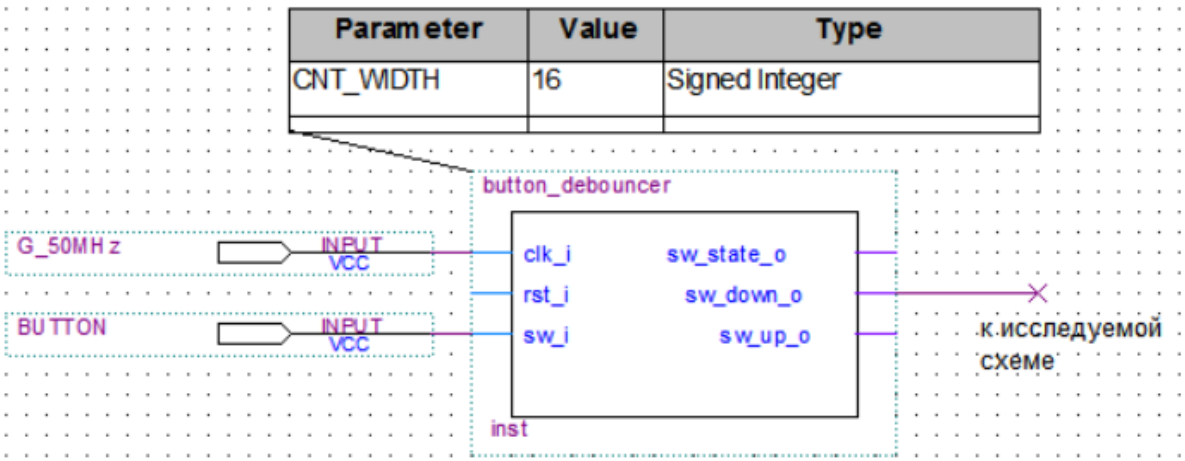


Рисунок В.2 - Приклад підключення схеми придушення брязкоту контактів

ДОДАТОК В

ОПИС МІКРОСХЕМИ SN74193 (К1533ІЕ7)

Мікросхема SN74193 (КР1533ІЕ7) являє собою двійковий чотирьох розрядний реверсивний лічильник синхронного типу.

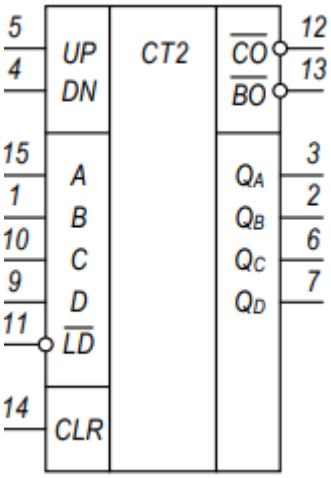


Рисунок В.1 - Умовно-графічне позначення мікросхеми SN74193(КР1533ІЕ7)

Таблиця В.1

Призначення виводів мікросхеми SN74193 (КР1533ІЕ7)

Номер вивода	Ім'я вивода	Функція вивода
1	B	Вхід інформаційний B
2	Q_B	Вихід B
3	Q_A	Вихід A
4	DN	Вхід «Зворотній рахунок»
5	UP	Вхід «Прямий рахунок»
6	Q_C	Вхід C
7	Q_D	Вхід D
8	GND	Спільний вивід
9	D	Вхід інформаційний D
10	C	Вхід інформаційний C
11	$\overline{LD}$	Вхід стробірованія попереднім записом

12	$\overline{CO}$	Вихід «Прямий перенос»
13	$\overline{BO}$	Вихід «Оборотній перенос»
14	CLR	Вхід скидання
15	A	Вхід інформаційний А
16	VCC	Вивід питання

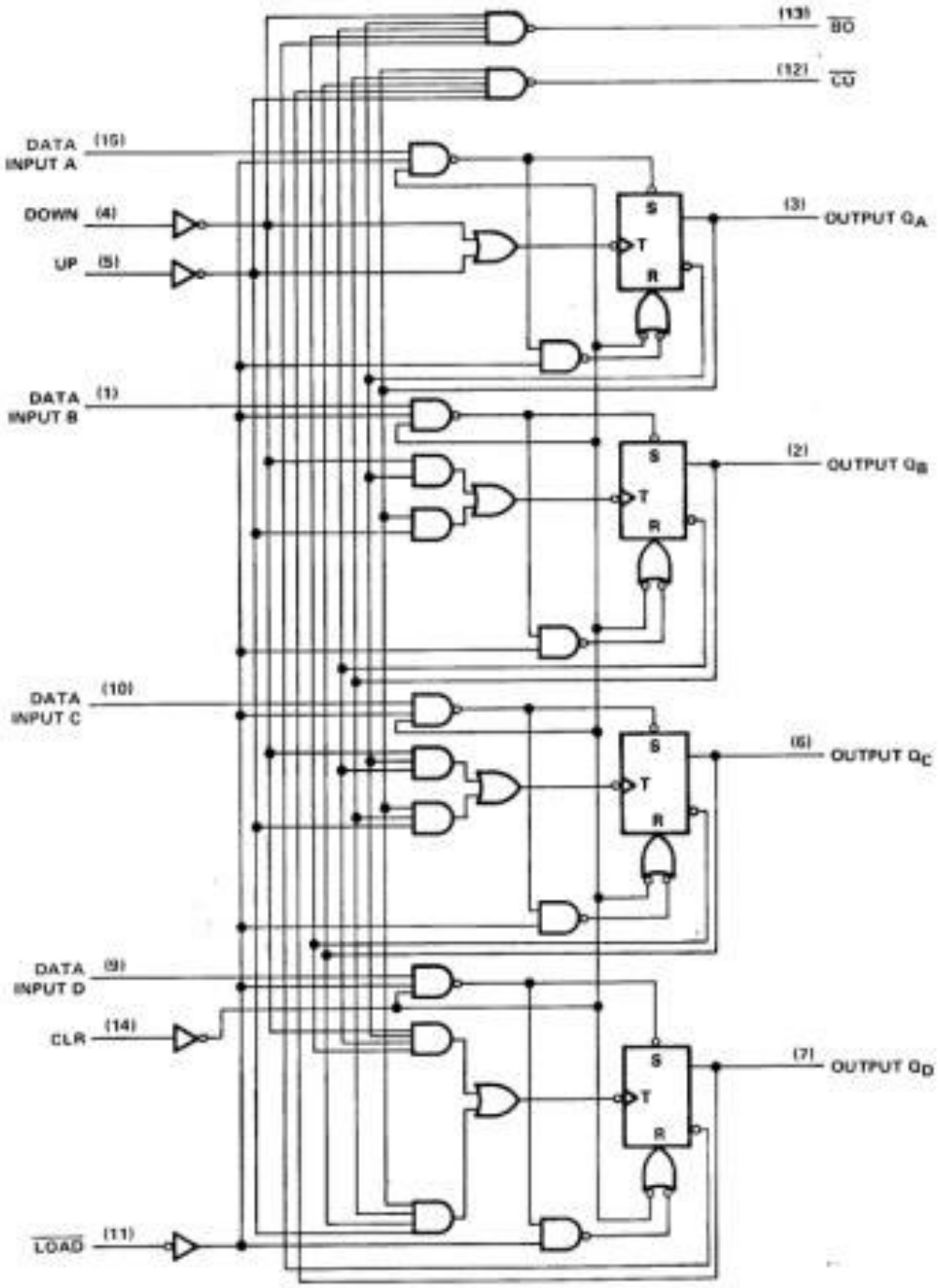


Рисунок В.2 - Функціональна схема мікросхеми SN74192 (КР1533ІЕ7)

Позитивний імпульс напруги по входу *CLR* встановлює виходи лічильника в початковий стан - рівень «логічного 0»на рахункових виходах. Для попередньої установки лічильника впевний стан необхідно на інформаційні входи подати відповідні рівні, а на вхід стробування попередньо котельної записи подати негативний імпульс напруги. Для здійснення прямого рахунку на вхід *DN* подається високий рівень напруги, а на вхід прямого рахунку *UP* - позитивні імпульси. Рахунок буде вестися від того числа, яке було попередньо записано в лічильник. Після заповнення лічильника виходи встановлюються в стан високого рівня, а на виході прямого перенесення з'явиться негативний імпульс перенесення рахунки в старший розряд. Аналогічно лічильник працює в режимі зворотного рахунку.

Таблиця В.2

Режими роботи мікросхеми SN74193 (КР1533ІЕ7)

Режим роботи	**Вхід**			
	CLR	$\overline{LD}$	UP	DN
Скидання виходів в лог. 0	1	X	X	X
Запис інформації	0	0	X	X
Неактивний стан	0	1	1	1
Рахунок прямий	0	1	⌐¬	1
Рахунок зворотній	0	1	1	⌐¬